Learning Representations for Log Data in Cybersecurity

Ignacio Arnaldo[1], Alfredo Cuesta-Infante[2], Ankit Arun[1], Mei Lam[1], Costas Bassias[1], and Kalyan Veeramachaneni[3]

[1] PatternEx Inc, San Jose, CA, USA,
iarnaldo@patternex.com,
[2] Universidad Rey Juan Carlos, Madrid, Spain
alfredo.cuesta@urjc.es
[3] MIT, Cambridge, MA, USA
kalyan@csail.mit.edu,

Abstract. We introduce a framework for exploring and learning representations of log data generated by enterprise-grade security devices with the goal of detecting advanced persistent threats (APTs) spanning over several weeks. The presented framework uses a divide-and-conquer strategy combining behavioral analytics, time series modeling and representation learning algorithms to model large volumes of data. In addition, given that we have access to human-engineered features, we analyze the capability of a series of representation learning algorithms to complement human-engineered features in a variety of classification approaches. We demonstrate the approach with a novel dataset extracted from 3 billion log lines generated at an enterprise network boundaries with reported command and control communications. The presented results validate our approach, achieving an area under the ROC curve of 0.943 and 95 true positives out of the Top 100 ranked instances on the test data set.

Keywords: Representation learning, Deep learning, Feature discovery, Cybersecurity, Command and Control detection, Malware detection

1 Introduction

This paper addresses two goals. First, it proposes methods to develop models from *log* and/or *relational* data *via* deep learning. Second, it applies these methods to a cybersecurity application.

Consider advanced persistent threats (APTs). These attacks are characterized by a series of steps: infection/compromise, exploitation, command and control, lateral movement, and data exfiltration [1,19]. In this paper, we focus on the detection of the "command and control," step, i.e. the mechanisms used to maintain a communication channel between a compromised host inside the targeted organization and a remote server controlled by the attacker. Although this phase of the attack can span weeks or months, its detection requires significant sophistication. Savvy attackers minimize their footprints by combining

active and stealthy phases, and establish communication channels via unblocked services and protocols, therefore blending in with legitimate traffic.

When *log* data is analyzed over a period of several weeks, these communications exhibit distinctive network profiles [9]. In particular, compromised machines will periodically attempt to communicate with remote server(s), and repeatedly establish lightweight connections through which they receive new instructions. During a minor fraction of these connections, the compromised machine will download a larger amount of data, which corresponds to a software update [19]. The frequency and network profile of these connections will depend on the particular malware family or exploit involved in the attack [15]. Despite these aforementioned observations, most machine learning-based detection techniques only analyze individual connections (see [21] and therein). Given the large volume of data, and the number of connections that must be monitored and analyzed, it is a challenge to identify behavioral patterns over multiple weeks of data. [4]

This example application identifies two pressing needs that could be addressed by deep learning. They are: (1) the development of automated methods to identify patterns, a.k.a *features* by processing data collected over long periods of time and (2) the identification of patterns that can deliver highly accurate detection capability. In recent years, data scientists have made tremendous strides in developing deep learning-based models for problems involving language (which use text as data) and vision (which use images as data). Deep learning has turned out to be so powerful in these two domains because it is able to produce highly accurate models by working with raw text or images directly, without requiring humans to transform this data into features. At its core, almost all deep learning models use multi-layered neural networks, which expect numeric inputs. To generate these inputs, images or text are transformed into numerical representations (often designed by humans).

In order to develop similar solutions for *log* or *relational* data, our first goal is to identify ways to process and generate numerical representations of this type of data. To maximize both quality and efficiency, this step requires a compromise between the amount of human knowledge we incorporate into developing these representations, *vs.* how much we exploit the ability of deep neural networks to automatically generate them. In this paper, we present multiple ways *log* data can be represented, and show how deep learning can be applied to these representations. Our contributions through this paper are:

– **Deep learning for log/relational data**: We present multiple ways to represent log/relational data, and 4 different deep learning models that could be applied to these representations. To the best of our knowledge, we are the first to elucidate steps for generating deep learning models from a *relational* dataset.

[4] Depending on an organizations size and level of activity, devices such as next-generation firewalls can generate up to 1TB of log data and involve tens of millions of entities on a daily basis.

– **Applying deep learning to cybersecurity applications**: We apply these methods to deliver models for two real world cybersecurity applications.
– **Comparing to human-driven data processing**: We demonstrate the efficacy of the deep learning models when compared to simple aggregations generated by human-defined standard database operations.

The rest of this paper is organized as follows. Section 2 describes the steps required to process raw logs and obtain data representation suitable for deep learning. Section 3 introduces the collection of deep learning techniques. We build upon these techniques and augment them with human-generated features for better discovery in Section 4. The experimental work and results are presented in Section 5 and Section 6. Section 7 presents the related work, and we conclude in Section 8.

2 Data transformations and representations

In this section, we describe a generic sequence of data transformation steps human data scientists can take to derive *features* from timestamped log data. With these transformations, we identify the data representations that can be fed into a deep learning technology.

Rep 1: Raw logs In a nutshell, *logs* are files that register a time sequence of events associated with entities in a monitored system[5]. Logs are generated in a variety of formats: `json`, `comma-separated-values`, `key-value` pairs, and `event logs`. (Event types are assigned unique identifiers and described with a varying number of fields).

Rep 2: Structured representation The first step entails parsing these *logs* to identify entities (e.g, IP addresses, users, customers) relationships between entities (e.g., one-to-many, many-to-many), timestamps, data types and other relevant information about the relational structure of the data. The data is then stored either in the relational data model (`database`) or *as-is*, and the relational model is used to process the data as needed. Either way, in this step, humans either come up with a relational model using their prior knowledge of how data is collected and what it means, or acquire this knowledge by exploring data.

Rep 3: Per entity, event-driven time series: Once the relational structure is identified, a temporal sequence of events associated with a particular instance of an entity is extracted. For example, we may identify all the events associated with a particular IP address. These events are usually irregular in time, and each event is described with a number of data fields. For example, a sequence of network connection events associated with an IP address is described using ports, protocols, the number of bytes sent and received, etc.

[5] In enterprises today, logs are generated by network devices, endpoints, and user authentication servers, as well as by a myriad of applications. Each device registers a certain kind of activity, and outputs different information. Note that even devices belonging to the same category (eg. network devices such as firewalls) report different information and use a different format depending on the vendor and version.

Rep 4: Per entity, aggregated, regular time series: The next step involves defining a time interval and performing simple aggregations for each irregular time series.[6] For example, for any given IP address, we can average the number of bytes sent and received per connection over a time interval of an hour. The output of this step is a regular multivariate time series for each entity-instance. The resulting data representation can be viewed as a multidimensional array $D \in \mathbb{R}^{n \times t \times p}$ where n is the number of entity-instances, t is the number of time steps, and p is the number of aggregations.[7]

Rep 5: Entity-feature matrix: This last step consists of generating an entity-feature matrix, in which each row corresponds to an instance of the entity and each column is a feature. This can be directly generated from Rep 3 through a process known as "*feature engineering*" or Rep 4. Given a multivariate time series $D \in \mathbb{R}^{n \times t \times p}$, the simplest way to generate this representation is to "flatten" the data, resulting in a $n \times (t \times p)$ feature matrix. A common alternative is to perform another aggregation step, this time on top of the regular time series. In the latter case, the result is a $n \times p'$ matrix, where p' is the number of second-level aggregations.

Data representations amenable for deep learning One important benefit of deep learning models is their potential to alleviate the feature engineering bottleneck. Below we consider the nuances of the application of deep learning models to different representations.

1. Input data must be separated into independent examples, much like images. Thus, it is necessary to identify the relational structure, and to separate data by entity-instances. Automation of this step is possible, but is beyond the scope of this paper.
2. Deep learning techniques can be applied to the third (per entity event-driven time series), fourth (aggregated regular time series), and fifth (entity-feature matrix) representations.
3. Note that while deep learning models can be applied to entity-feature matrices (last representation), we consider that this approach does not leverage their potential for feature discovery, since multiple levels of aggregations are defined by humans.
4. In this paper, we leverage deep learning techniques to learn features on regular aggregated time series (Rep 4.).

3 Learning representations using deep neural networks

We describe 4 different methods suitable for learning representations out of Rep 4. 1) Feed-forward neural networks, 2) Convolutional networks, 3) Recurrent

[6] For numeric fields, aggregations include *minimum*, *maximum*, *average*, and *standard deviation*; for categorical values, common aggregations are *count_distinct* and *mode*.

[7] For example, if we consider a dataset spanning over 10 days with $n = 1000$ entity instances, a time step $t = 1$ day, and $p = 20$ aggregations, the result of this step would be a $1000 \times 10 \times 20$ array.

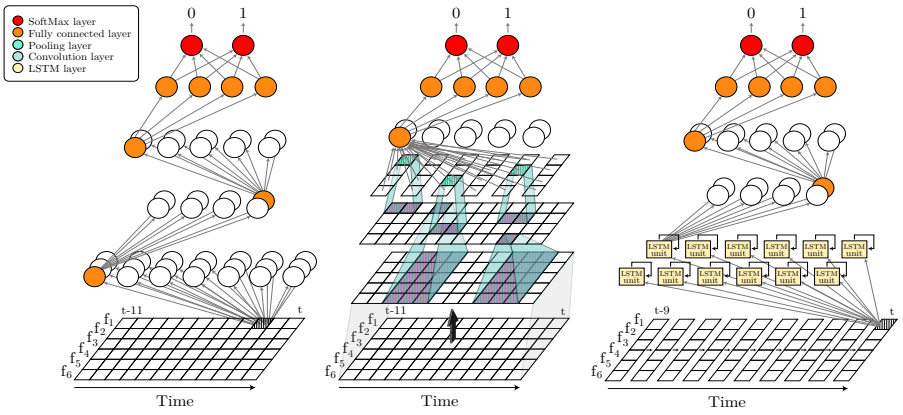

Fig. 1: FFNN-based (left), CNN-based (center), and LSTM-based (right) time series classifiers. For clarity, not all the connections are shown.

neural networks with LSTMs, and 4) Autoencoder + random forest pipeline. The first three approaches can be categorized as methods for supervised *feature extraction* or *feature learning* and the last method (without random forest) can be categorized as an an unsupervised approach to *feature learning*.

3.1 Feed-forward neural networks

Feed-forward neural networks (FFNNs) are composed of one or more layers of nodes. The input layer consists of $p \times d$ neurons (one for each value in the input data), while the output layer is composed of m nodes, where m is the number of classes in the data. Intermediate layers are composed of an arbitrary number of nodes, with each layer fully connected to the next one. Figure 1 (left) shows a FFNN trained to classify multivariate time-series.

3.2 Convolutional networks

Convolutional networks (CNNs or ConvNets) are FFNNs with special connection patterns (see Figure 1), and have been widely applied for image and video recognition. At the core of CNNs are convolutional filters or *kernels*. These filters are trained to identify patterns in reduced regions of the input data (small shapes in the case of images, or patterns in consecutive data points in the case of time series). CNNs are composed of an arbitrary number of such filters, and are therefore capable of identifying a wide variety of low-level patterns in the data. The same set of filters is applied across all the input data, and for each region of the input data where they are applied, the filter generates an output value that indicates how similar the input region is to the filtered pattern. The output of the convolutional layer is generally fed to a pooling layer, which applies a local maximum operation. Intuitively, this operation provides robustness to determine

whether a pattern exists in a region of the input data, independent of its exact location. The outputs of the last convolutional/pooling layers are fed to a fully connected feed-forward neural network. As with standard FFNNs, the final layer is composed of m nodes, where m is the number of classes in the data.

By stacking several layers of convolutional filters and pooling layers, CNNs can identify patterns involving larger regions of the input data. This is a clear example of a "deep" architecture, where lower layers learn to detect building blocks of the input data, while the last layers detect higher-level patterns. It is important to stress that all the parameters (weights) in CNNs are learned during the training process. That is, the networks learns to identify the local patterns that ultimately help them to discriminate between data categories.

In the case of multivariate time-series data, CNNs can exploit locality to learn temporal patterns across one or more variables. Note however, that the relative position of features is generally arbitrary (adjacent features are not necessarily related). This observation motivates the use of convolutional filters of width= 1; that is, filters that learn patterns in each feature independently. Another valid possibility explored in this paper considers filters of width= p, where p is the total number of input features. In this last case, the network will learn filters or patterns involving all the features.

3.3 Recurrent neural networks with LSTMs

Long short-term memory networks (LSTMs) are a special case of recurrent neural networks first introduced in [11]. The main characteristic of these architectures is the use of LSTM cells. LSTM cells maintain a state, and generate an output given a new input and their current state. Several variants of LSTM cells have been proposed, we use the LSTM variant introduced by [17].

Right panel in Figure 1 shows a high-level representation of a LSTM architecture. The input data is a time-ordered array that is fed sequentially to the network. At each time step, the LSTM cells update their state and produce an output that is related both with the long-term and short-term (i.e. recent) inputs. The final output of the LSTM architecture is generated after propagating all the input sequence through the network.

LSTMs architectures are a solution to the vanishing and exploding gradient; they are said to be superior to recurrent neural networks and Hidden Markov Models to model time series with arbitrarily large time gaps between important events. With respect to FFNNs and CNNs, their main potential advantage is that inputs to LSTM architectures are sequences of arbitrary length, therefore enabling us to train and reuse a single model with time series of different lengths. These two characteristics of LSTMs are particularly relevant for information security analytics, where the goal is to detect attacks that are generally implemented in steps spread over time, and where modeled entities exhibit very different levels of activity, therefore generating time series of varying length.

3.4 Autoencoder + Random Forest pipeline

Autoencoders are multi-layer feed-forward neural networks. The input and output layers have the same number of nodes, while intermediate layers are composed of a reduced number of nodes. We consider autoencoders that are composed of three hidden layers. The first and third hidden layers count $p/2$ neurons, while the second, central layer is composed of $p/4$ neurons, where p is the dimensionality of the data. The tan-sigmoid transfer function is used as an activation function across the network. The network is trained to learn identity-mapping from inputs to outputs. The mapping from inputs to intermediate layers compresses the data, effectively reducing its dimensionality. Once the network is trained, we can compress the data by feeding the original data to the network, and retrieving the output generated at the central layer of the autoencoder. The output of the central layer is then fed to a random forest classifier.

4 Combining human defined and learnt features

In this section, we determine whether feature discovery techniques are contributing to improvements in classification accuracy. We do this by separating the aggregated values corresponding to last time step, generated as part of REP 4, from the historic data (previous time steps), and applying the feature discovery methods only to the historic data. All the presented techniques are extensions of the methods described in Section 3.

Let $D^i = D^i_{hist} \cup D^i_{last}$ be the multivariate time series associated to entity i, and let d be the number of time steps in the series. Therefore, D^i_{last} is the aggregated time series vector corresponding to the last time step data and D^i_{hist} is the time series composed of the previous $(d-1)$ vectors. In our case, the time unit is 1 day, and we consider $d = 28$ time steps. We introduce a pipeline where:

- Deep learning methods learn a set of "time series features" from D_{hist},
- These learned features are concatenated with D_{last}.
- The combination of learned "time series features" and D_{last} is fed into a random forest classifier.

This way, feature discovery techniques learn a set of "time series features" while the final predictions are generated by interpretable models. By analyzing the grown decision trees, we determine the relative importance of D_{last} and the automatically discovered features. In the following, we describe unsupervised and supervised techniques to discover new features from historic data.

4.1 Extension of dimensionality reduction methods (RNN)

Given a time series dataset $D = D_{hist} \cup D_{last}$, we first apply a dimensionality reduction technique to D_{hist} (historic feature values). The outputs of the dimensionality reduction are combined with the last time step's feature vector and fed into a random forest as depicted in Figure 2. We use the same dimensionality reduction technique explained in Section 3, namely RNNs or autoencoders. In the following, we refer to this extension as 'RNN + RF ext'.

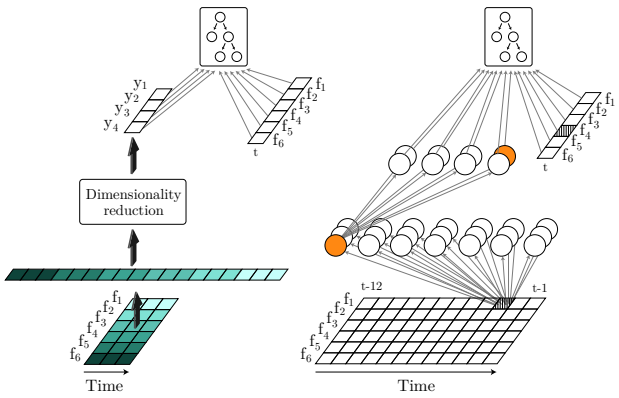

Fig. 2: Dimensionality reduction and random forest pipeline (left), and FFNN-based model used as time-series feature generator (right)

4.2 Extension of supervised deep learning (FFNN, CNN, LSTM)

We consider the models depicted in Figure 3. The designed models have two separate inputs: D_{hist} and D_{last}. While D_{hist} undergoes a series of nonlinear transformations in the left layers of the network, D_{last} is directly connected to the last layers of the network. With this design, we expect to force the network to learn features from D_{hist} that are complementary to D_{last}. Once trained, these models can be used in two fashions: 1) as standalone models used to predict on unseen data, 2) as "feature generators" used to extract features for unseen data. In this paper we adopt the second strategy, and feed the extracted features into a random forest classifier. We illustrate this strategy in Figure 2 (right).

We now detail the steps involved both in model training and deployment using these "feature generators". At training time, we proceed as follows:

1. Train the models in Figure 3 via backpropagation using the dataset D.
2. Once the model is trained, propagate D through the network and retrieve the outputs generated at the last layer of the left section. Note that the output D_{ts} will be a matrix of shape $n \times q$, where n is the number of examples and q is the number of learned features.
3. Concatenate D_{last} and D_{ts} to obtain D_{conc}, a new dataset with shape $n \times (p + q)$. Note that p is the number of human-engineered features.
4. Train a decision tree classifier with D_{conc}

To predict on unseen data D', we proceed as follows:

1. Propagate D' through the network and retrieve the outputs generated at the last layer of the left section of the network. As in the training scenario, the output D'_{ts} will be a matrix of shape $n \times q$
2. Concatenate D'_{last} and D'_{ts} to obtain D'_{conc}
3. Feed D'_{conc} to the trained random forest and generate predictions.

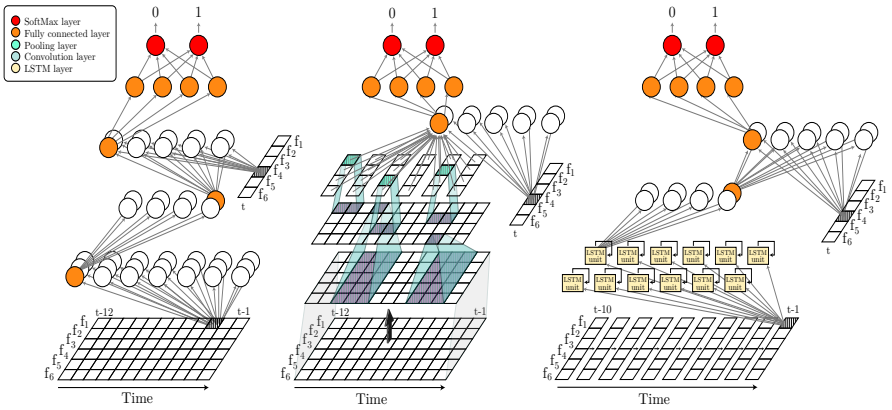

Fig. 3: FFNN-based (left), CNN-based (center), and LSTM-based (right) models designed to learn time series features. These models present a novel structure that enables to complement a set of existing features with new features learned from historic data.

In the following, we refer to these method extensions as 'FFNN + RF ext', 'CNN + RF ext', and 'LSTM + RF ext'.

5 Experimental Work

This section describes the two datasets studied in this paper, as well as the parameterization of the models introduced in previous sections.

5.1 Real-world command and control detection dataset

We consider two months worth of logs generated by an enterprise next generation firewall and target the detection of command and control. These log files register approximately 50 million log lines and 150K active entities daily, summing to a total of 3 billion log lines and 12 million analyzed entities.

Extracting daily features: In this step, we extract 32 features, describing the activity of each entity within a 24-hour time window. The features capture information about the number of connections, the bytes sent and received per connection, the packets sent and received per connection, the duration of the connections, and the intervals between connections, as well as information about relevant ports, applications and protocols, and alerts raised by the firewall.

Labeling the dataset: To label the dataset, we use a combination of outlier analysis and validation through VirusTotal's [4] threat intelligence. We perform outlier analysis (see methods in [23]) on the feature data on a daily basis, and investigate the top outliers using VirusTotal. VirusTotal provides the latest files detected by at least one antivirus program that communicates with a given IP

address when executed in a sandboxed environment. In addition, it provides the fraction of considered antivirus software that reported the file. We consider an IP address to be malicious if at least 5 different antivirus programs or scanners report that malicious files are communicating with that address.

It is important to note that this intelligence-based labeling process is noisy, not only because antivirus programs themselves might present false positives, but because malicious files might communicate with IP addresses for purposes other than command and control. In fact, creating and labeling a real-world dataset is challenging in itself, both because the labeling must be performed by human analysts, a scarce resource with very limited bandwidth, and because the context required to determine whether a particular entity is involved in an attack is often missing. This severely limits the number of labeled attack examples available for offline modeling and experimentation.

Building a control dataset: We preserve all the attack examples, and subsample 1% of the remaining entities, which are considered benign. The result is a dataset composed of $89K$ examples. The data pertaining to the first month ($53K$ entities) is used to train the models, while data from the second month ($36K$ entities) is used for testing. It is worth noting that, although we analyze a subsampled dataset, malicious activities remain a minor fraction (0.56%) of the examples. This results in an extreme class imbalance, which increases the difficulty of the detection problem.

From daily features to multi-week time series: For each sampled entity, we build a multivariate time series in which the time step is a day, the length of the series is $d = 28$ days, and the activity at each time step is described with 32 features. Therefore, our dataset can be viewed as a $89K \times 28 \times 32$ array (num. examples $\times$ time steps $\times$ features).

5.2 ISCX Botnet Dataset

Introduced in 2014, the ISCX Botnet dataset [5] is a comprehensive dataset released in packet capture (pcap) format which contains activity traces of 16 types of botnets, as well as legitimate traffic. To build the dataset, the authors combined traces extracted from the ISOT botnet detection dataset [27], the ISCX 2012 IDS dataset [18], and traffic generated by the Malware Capture Facility Project [3]. The botnet traffic is either captured by honeypots, or through executing the bot binaries in safe environments. Table 1 summarizes the characteristics of the data. It is important to note that the dataset is divided into a training (4.9GB) and testing set (2.0GB), where the training split includes traffic generated by 7 botnet types, while the testing set contains traffic generated by 16 botnet types. This way, the authors propose a challenging dataset to evaluate whether models that have been trained to detect a reduced set of botnets can accurately detect unseen botnets. In their best-performing effort, the authors report a detection (true positive) rate of 75% and a false positive rate of 2.3%.

From pcap to flow features: We use FlowMeter [8], a network traffic flow generator, to separate the packet capture data into individual flows. FlowMeter aggregates flows on the basis of the 5-tuple set (Source IP, Source Port, Destination IP, Destination Port, Protocol) and a timeout parameter. Each flow is

Split	#Flows	#Src IPs	#Dst IPs	#Src/Dst IPs	#Flow TS	#Mal. TS	#Ben. TS
Training	356160	7355	40502	57321	65737	38485	27252
Testing	309206	6392	17338	28657	36532	13480	23052

Table 1: Characteristics of the ISCX 2014 Botnet Dataset

Method	#discov. features	#layers	Training algorithm
PCA + RF	16	-	-
RNN + RF	8	3 (16-8-16)	Adam
FFNN	16	3 (16-16-16)	Stoch. grad. descent
CNN	16	2(conv+pool) + 1 fully conn.	Adam
LSTM	16	1 layer with 100 LSTM cells	Adam
RNN + RF ext	8	3 (16-8-16)	Adam
PCA + RF ext	16	-	-
FFNN + RF ext	16	3 (16-16-16)	Stoch. grad. descent
CNN + RF ext	16	2(conv+pool) + 1 fully conn.	Adam
LSTM + RF + int	16	1 layer with 100 LSTM cells	Adam

Table 2: Description and number of features generated by the compared models

described with the following 23 features: *Duration*, *Bytes per second*, *Packets per second*, *Min/Max/Avg/Std packet inter-arrival times*, *Min/Max/Avg/Std inter-arrival times of sent packets*, *Min/Max/Avg/Std inter-arrival times of received packets*, *Min/Max/Avg/Std active time*, and *Min/Max/Avg/Std idle time*.

Labeling the dataset: The dataset includes a list of malicious IPs and their corresponding botnet types. In some cases, individual IPs are reported, but in others, the authors report source and destination IPs as a pair. Therefore, we label as malicious all flows that include one of the individually listed IPs (either as source or destination), and all flows where both the source and destination IPs match a reported pair. All remaining flows are considered benign. Although the authors report the botnet type associated with the malicious IPs, we approach the problem as a binary classification problem (malicious vs benign).

Flow features to time series of flows: We first aggregate all the flows that involve the same pair of source and destination IPs (independently of ports and protocols). Thus, for each pair of source/destination IPs, we obtain a $t \times p$ matrix, where t represents the number of flows, and $p = 23$ represents the number of features. For symmetry with the real-world dataset, we split time series into segments of (at most) 28 flows. Note that this last step is only applied when the pair of source/destination IPs presents more than 28 flows. This way, the preprocessing of the training split results in a $65737 \times 28 \times 23$ array (num. examples $\times$ flows $\times$ features), while the testing split results in a $36532 \times 28 \times 23$ array.

5.3 Model implementation, training, and validation

We compare the models proposed in this paper against random forests [6] and against a pipeline composed of a dimensionality reduction step performed with PCA followed by a random forest classifier. Note that we consider data composed of n examples, p features, and d days. In order to apply these approaches, the data is flattened to obtain a feature matrix with n examples and $p \times d$ features. The resulting entity-feature matrix is suitable for training random forests. In the case of the PCA + Random Forest pipeline, the data is projected to the principal component space using the top j principal components, and the projected data is fed to a random forest classifier. Its extended counterpart is referred to as 'PCA + RF ext' and is analogous to the RNN-based method explained in Section 4.1.

Table 2 shows the details of the implementation and training of the models compared in this paper. To enable a fair comparison with methods such as random forests or PCA, we did not parameter-tune any of the neural network-based methods (FNN, CNN, LSTM, Autoencoders (RNN)). While this can lead to a poor model parametrization, we are interested in these methods out-of-the-box performance, since it is a better performance proxy for how well they will detect malicious behaviors other than command and control.

6 Results

In this section, we compare the detection performance of the learning methods introduced in this paper on the real-world command and control dataset and on the ISCX 2014 botnet dataset. We also analyze the importance of automatically-discovered features.

6.1 Real-world command and control dataset

Table 3 shows the AUROC and true positives in the top 100 compared methods when evaluated on unseen data.

Effect of longer time span data: Our first observation is that the AUROC achieved using 1 day of data reaches 0.923 for *RF* and 0.928 for *PCA + RF*. However, if we use more days for training, the performance of these two methods degrades. This degradation is noteworthy since we do not know beforehand the length of the time necessary for the successful detection malicious behaviors.

Did augmentation help?: On average, the AUROC and number of TP in the Top 100 for the extended methods that try to complement human generated features (i.e methods labeled with *ext*) is higher than the ones that don't. Note that, by design, the methods CNN, LSTM, PCA + RF ext, RNN + RF ext, FFNN + RF ext, CNN + RF ext, and LSTM + RF ext require more than one day of data; therefore, for those methods, we do not present performance metrics for the one-day case. We also notice that the performance of these methods does not degrade as we increase the time span.

	AUROC				True Positives in Top 100			
Method	1 day	7 day	14 days	28 days	1 days	7 days	14 days	28 days
RF	0.923	0.895	0.881	0.883	95	84	89	82
PCA + RF	0.928	0.830	0.816	0.867	86	66	68	74
RNN + FR	0.814	0.747	0.686	0.701	37	35	4	19
FFNN	0.906	0.840	0.829	0.869	7	0	0	0
CNN	-	0.901	0.718	0.873	-	0	1	4
LSTM	-	0.898	0.877	0.869	-	8	26	31
PCA + RF ext	-	0.920	0.927	0.943	-	89	92	87
RNN + RF ext	-	0.747	0.678	0.756	-	9	30	3
FFNN + RF ext	-	0.929	0.888	0.912	-	92	93	92
CNN + RF ext	-	0.936	0.876	0.837	-	95	89	74
LSTM + RF ext	-	0.904	0.914	0.923	-	88	89	89

Table 3: AUROC and true positives in the top 100 of the compared methods when evaluated on unseen data. Data sets are represented by their time span (1, 7, 14 and 28 days).

The best and the worst: The best AUROC is achieved using *PCA + RF ext* with 28 days of data, and using *CNN + RF ext* with 7 days of data. These models present AUROCs of 0.943 and 0.936 respectively when evaluated on unseen data. However, this is only marginally better than the 0.923 baseline AUROC obtained with a random forest classifier using one day of data. In particular, our results show that the use of RNN + RF (autoencoders) achieves the worst detection performance since it is unable to either detect attacks or discover new features.
Key findings: The goal of our exploration was to examine, how these methods perform "out-of-box". For the real world data set, based on our results, we cannot conclusively say whether the new learning methods helped. We also posit that:

- perhaps the information present in the last day's features is enough to accurately detect command and control communications.
- the performance of those methods using FFNN, CNN, LSTM, and RNN (autoencoders) can be improved via parameter tuning.

6.2 ISCX 2014 Botnet Dataset

Method comparison: Given that the training and testing splits contain traces of different botnets, we perform two series of experiments. First, we compute the 10-fold cross-validation AUROC on the training set for the models being compared (left section of Table 4). This setup allows us to compare the models' capacity to identify known botnets on unseen data. Second, we compute the testing set AUROC to analyze the models' capacity to identify unseen botnets (right section of Table 4). The resulting detection rates are in accordance with the results reported in [5], where the authors present very high accuracies in cases where traces of the same botnets are included in the training and testing splits, while the detection rates on unseen botnets drop significantly. For instance, the AUROC of the random forest trained on individual flows drops from 0.991 to

	CV AUROC Training Set				AUROC Training/Testing Set			
Method	1 flow	7 flows	14 flows	28 flows	1 flow	7 flows	14 flows	28 flows
RF	0.991	0.997	0.997	0.997	0.768	0.753	0.766	0.748
PCA + RF	0.990	0.992	0.992	0.992	0.769	0.753	0.743	0.774
RNN + FR	0.975	0.971	0.965	0.955	0.746	0.747	0.741	0.641
FFNN	0.905	0.947	0.947	0.948	0.724	0.713	0.751	0.744
CNN	-	0.737	0.632	0.620	-	0.644	0.633	0.449
LSTM	-	0.907	0.903	0.899	-	0.624	0.744	0.542
PCA + RF ext	-	0.997	0.830	0.832	-	0.788	0.802	0.811
RNN + RF ext	-	0.995	0.809	0.805	-	0.715	0.701	0.728
FFNN + RF ext	-	0.978	0.949	0.986	-	0.731	0.739	0.788
CNN + RF ext	-	0.929	0.936	0.849	-	0.748	0.747	0.759
LSTM + RF ext	-	0.904	0.932	0.901	-	0.672	0.779	0.808

Table 4: Cross-validation AUROC on the training split of the ISCX training split, and AUROC on the testing split of the compared methods. Data sets are represented by the number of considered flows (1, 7, 14 and 28 flows).

0.768. As stated in [5], this performance drop shows that the trained models do not generalize well when it comes to accurately detecting unseen botnets.

Detecting previously seen botnets: With the exception of the CNN method, all methods achieve high cross-validation detection metrics. In particular, when 7 consecutive flows are modeled, the AUROCs range from 0.904 to 0.997. The best cross-validation results are achieved by the random forest and PCA + Random Forest methods, which achieve AUROCs of 0.997 and 0.992 respectively. In both cases, considering multiple flows yields incremental benefits (from 0.991 to 0.997, and from 0.990 to 0.992). However, in general, considering multiple flows does not systematically improve detection. While it yields improvements for RF, PCA+RF, FFNN, FFNN+RF ext, CNN+RF ext, and LSTM+RF ext, it results in performance decreases for RNN+RF, CNN, LSTM, PCA+RF ext, and RNN+RF ext.

Did augmentation help?: The AUROCs of methods that complement human-engineered features (i.e methods labeled with *ext*) are higher than those for the complementary subset for FFNN, CNN, and LSTM models, and lower for PCA and RNN.

Detecting previously unseen botnets: When it comes to detecting unseen botnets, the best AUROCs are achieved by the models PCA + RF ext (0.811) and LSTM + RF ext (0.808), which in both cases model segments of 28 consecutive flows. This represents an improvement of 5.60% and 5.21% with respect to the baseline random forest trained with individual flows (0.768). In this case, the AUROC of all the extended methods that complement human-engineered features (i.e methods labeled with *ext*) is higher than the complementary subset.

Feature Analysis: We analyze the features discovered with the models *PCA + RF ext* and *LSTM + RF ext* using 28 days of data (see Section 4.1 and

Method	Aggregated Feature Importance							
	All examples		7 flows or more		14 flows or more		28 flows	
	Human	Auto	Human	Auto	Human	Auto	Human	Auto
PCA + RF ext	86.6%	13.4%	54.1%	45.9%	52.7%	47.3%	51.2%	48.8%
LSTM + RF ext	85.7%	14.4%	54.6%	45.4%	50.6%	49.4%	52.3%	47.7%

Table 5: Aggregated importance of human-engineered and discovered features.

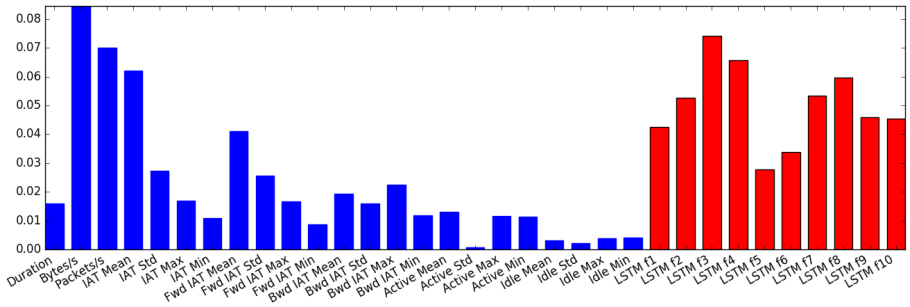

Fig. 4: Feature importance as determined by a random forest classifier of human-engineered features (blue) and automatically discovered features (red) with a LSTM-based model. The aggregated importance of human-engineered features is 50.6%, while that of discovered features is 49.4%. Only pairs of source/dest IPs with 14 or more flows are considered for the analysis presented in this figure.

Section 4.2). These models are chosen for analysis because they present the highest AUROCs (0.811 and 0.808 when evaluated on unseen data).

Given that the training and testing sets contain traces generated by different botnets, we merge the two splits, obtaining a single dataset composed of 102246 examples (65734 train + 36512 test). Also, since many of the modeled pairs of source and destination IPs present a reduced number of flows, we analyze feature importance over varying lengths of the time series. This way, we consider four different views of the data:

1. All pairs of source and destination IPs. This results in a dataset composed of 102246 examples, out of which 51946 are malicious.
2. Pairs of source and destination IPs presenting at least 7 flows. This results in 22093 examples, out of which 9093 are malicious
3. Pairs of source and destination IPs presenting at least 14 flows. This results in 18849 examples, out of which 8336 are malicious
4. Pairs of source and destination IPs presenting at least 28 flows. This results in 16414 examples, out of which 7529 are malicious.

Table 5 reports the sum of the importance of all human-engineered features, as well as the sum of the importance of all automatically discovered features. The results show that the discovered features are used by the classifier in all four

scenarios. The aggregated importance of learned features is 13.4% and 14.4% for PCA-based and LSTM-based features when all examples are considered. These low values are explained by the fact that most pairs of source/destination IPs present a single flow, and so the classifier relies on the individual flow features. However, the aggregated importance of discovered features increases as we consider examples composed of 7, 14, and 28 of flows. In particular, the importance of human and LSTM-based features is close to parity (50.6% vs. 49.4%) when examples composed of 14 or more flows are considered. This case is highlighted in Figure 4, in which we show the importance of the 23 original flow features (blue) and the 10 features (red) as determined by a random forest classifier learned with the LSTM model. The most important human-engineered features are *Bytes per second* (1st overall), *Packets per second* (3rd overall), and *Avg inter-arrival time* (5th overall). All LSTM features are used and considered important. Features LSTM-3, LSTM-4, and LSTM-8 are ranked 2nd, 4th, and 6th in overall importance.

7 Related Work

There is a large research community focused on addressing InfoSec use cases with machine learning [12]. The command and control detection problem, and botnet detection in particular, has been widely studied (see [21,10] and therein). Two key aspects differentiate this paper from existing work. First, most research initiatives consider publicly available datasets that are either synthetic or generated in controlled environments. Working with public datasets allows researchers to replicate reported methodologies and to compare results fairly. However, these datasets generally suffer from a lack of generality, realism, and representativeness [5], and results obtained using them do not necessarily transfer to real-world deployments. In this paper, we work with a dataset obtained over two months from a real-world system. (Obtaining representative, real-world datasets is a challenge in itself, and has been discussed in previous sections.)

Second, despite observations indicating that command and control communications exhibit distinctive network profiles when analyzed over long periods of time [9], most existing approaches model individual flows [21]. In [5], the authors suggest a potential improvement for modeling capabilities that includes multidimensional snapshots of network traffic, e.g., combining flow level features with pair level features (a pair of source and destination, no matter which port and protocol used). This corresponds to the multivariate time-series classification approaches introduced in this paper.

Classification methods for multivariate time series have been studied extensively. Xi *et al.* [26] compared several approaches, including hidden Markov models [13], dynamic time warping decision trees [16], and a fully connected feed-forward neural network [14]. Wang *et al.* [24] explore different methods for projecting time series data into 2D images. The authors then explore the use of convolutional networks to tackle several regression problems from the UCR repository [7].

While deep learning-based solutions have been used for problems involving computer vision and natural language processing, only a few examples exist in

the domain of information security. Staudemeyer *et al.* [20] explore the use of recurrent neural networks with LSTMs to tackle the intrusion detection problem on the 1999 KDD Cup dataset [2]. A recent work by Tuor *et al.* [22] explores deep learning methods for anomaly-based intrusion detection. There are also reported approaches that leverage LSTM-based models to differentiate the algorithmically-generated domains used for command and control from legitimate ones [25]. This paper is, to the best of our knowledge, the first paper to introduce a generic framework for discovering features from any set of time-stamped log data. Moreover, this is the first attempt to automatically discover features that complement existing human-engineered features.

8 Conclusions

In this paper, we have presented multiple ways to represent log/relational data, and 4 different deep learning models that can be applied to these representations. We apply these methods to deliver models for command and control detection on a large set of log files generated at enterprise network boundaries, in which attacks have been reported. We show that we can detect command and control over web traffic, achieving an area under the ROC curve of 0.943 and and 95 true positives out of the Top 100 ranked instances on the test data set. We also demonstrate that the features learned by deep learning models can augment simple aggregations generated by human-defined standard database operations.

References

1. Adversarial tactics, techniques & common knowledge, `https://attack.mitre.org`
2. KDD Cup 99, `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`
3. Malware capture facility project, `http://mcfp.weebly.com/`
4. VirusTotal, `https://www.virustotal.com`
5. Beigi, E.B., Jazi, H.H., Stakhanova, N., Ghorbani, A.A.: Towards effective feature selection in machine learning-based botnet detection approaches. In: 2014 IEEE Conference on Communications and Network Security. pp. 247–255 (2014)
6. Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001)
7. Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G.: The UCR time series classification archive (2015)
8. Draper-Gil, G., Lashkari, A.H., Mamun, M.S.I., Ghorbani, A.A.: Characterization of encrypted and VPN traffic using time-related features. In: Proceedings of the 2nd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,. pp. 407–414 (2016)
9. García, S., Uhlíř, V., Rehak, M.: Identifying and modeling botnet C&C behaviors. In: Proceedings of the 1st International Workshop on Agents and CyberSecurity. pp. 1:1–1:8. ACySE '14, ACM, New York, NY, USA (2014)
10. Garcia, S., Zunino, A., Campo, M.: Survey on network-based botnet detection methods. Security and Communication Networks 7(5), 878–903 (2014)
11. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. 9(8), 1735–1780 (Nov 1997)

12. Jiang, H., Nagra, J., Ahammad, P.: Sok: Applying machine learning in security-a survey. arXiv preprint arXiv:1611.03186 (2016)
13. Kim, S., Smyth, P., Luther, S.: Modeling waveform shapes with random effects segmental hidden markov models. In: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence. pp. 309–316. UAI '04, AUAI Press, Arlington, Virginia, United States (2004)
14. Nanopoulos, A., Alcock, R., Manolopoulos, Y.: Information processing and technology. chap. Feature-based Classification of Time-series Data, pp. 49–61. Nova Science Publishers, Inc., Commack, NY, USA (2001)
15. Plohmann, D., Yakdan, K., Klatt, M., Bader, J., Gerhards-Padilla, E.: A comprehensive measurement study of domain generating malware. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 263–278. USENIX Association, Austin, TX (2016)
16. Rodríguez, J.J., Alonso, C.J.: Interval and dynamic time warping-based decision trees. In: Proceedings of the 2004 ACM Symposium on Applied Computing. pp. 548–552. SAC '04, ACM, New York, NY, USA (2004)
17. Sak, H., Senior, A.W., Beaufays, F.: Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. CoRR abs/1402.1128 (2014)
18. Shiravi, A., Shiravi, H., Tavallaee, M., Ghorbani, A.A.: Toward developing a systematic approach to generate benchmark datasets for intrusion detection. computers & security 31(3), 357–374 (2012)
19. Sood, A., Enbody, R.: Targeted Cyber Attacks: Multi-staged Attacks Driven by Exploits and Malware. Syngress Publishing, 1st edn. (2014)
20. Staudemeyer, R.C., Omlin, C.W.: Evaluating performance of long short-term memory recurrent neural networks on intrusion detection data. In: Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference. pp. 218–224. SAICSIT '13, ACM, New York, NY, USA (2013)
21. Stevanovic, M., Pedersen, J.M.: On the use of machine learning for identifying botnet network traffic. Journal of Cyber Security and Mobility 4(3), 1–32 (2016)
22. Tuor, A., Kaplan, S., Hutchinson, B., Nichols, N., Robinson, S.: Deep learning for unsupervised insider threat detection in structured cybersecurity data streams (2017)
23. Veeramachaneni, K., Arnaldo, I., Korrapati, V., Bassias, C., Li, K.: AI^2: Training a big data machine to defend. In: 2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS). pp. 49–54 (2016)
24. Wang, Z., Oates, T.: Imaging time-series to improve classification and imputation. In: Proceedings of the 24th International Conference on Artificial Intelligence. pp. 3939–3945. IJCAI'15, AAAI Press (2015)
25. Woodbridge, J., Anderson, H.S., Ahuja, A., Grant, D.: Predicting domain generation algorithms with long short-term memory networks. arXiv preprint arXiv:1611.00791 (2016)
26. Xi, X., Keogh, E., Shelton, C., Wei, L., Ratanamahatana, C.A.: Fast time series classification using numerosity reduction. In: Proceedings of the 23rd International Conference on Machine Learning. pp. 1033–1040. ICML '06, ACM, New York, NY, USA (2006)
27. Zhao, D., Traore, I., Sayed, B., Lu, W., Saad, S., Ghorbani, A., Garant, D.: Botnet detection based on traffic behavior analysis and flow intervals. Computers & Security 39, 2–16 (2013)