BoTShark: A Deep Learning Approach for Botnet Traffic Detection

Sajad Homayoun, Marzieh Ahmadzadeh, Sattar Hashemi, Ali Dehghantanha, and Raouf Khayami

Abstract While botnets have been extensively studied, bot malware is constantly advancing and seeking to exploit new attack vectors and circumvent existing measures. Existing intrusion detection systems are unlikely to be effective countering advanced techniques deployed in recent botnets. This chapter proposes a deep learning-based botnet traffic analyser called Botnet Traffic Shark (BoTShark). BoTShark uses only network transactions and is independent of deep packet inspection technique; thus, avoiding inherent limitations such as the inability to deal with encrypted payloads. This also allows us to identify correlations between original features and extract new features in every layer of an Autoencoder or a Convolutional Neural Networks (CNNs) in a cascading manner. Moreover, we utilise a Softmax classifier as the predictor to detect malicious traffics efficiently.

Keywords Botnet · Intrusion detection · Network flows · Deep learning · Autoencoder · CNNs

S. Homayoun · M. Ahmadzadeh · R. Khayami
Department of Computer Engineering and Information Technology, Shiraz University of Technology, Shiraz, Iran
e-mail: S.Homayoun@sutech.ac.ir; Ahmadzadeh@sutech.ac.ir; Khayami@sutech.ac.ir

S. Hashemi
Department of Computer Engineering, Shiraz University, Shiraz, Iran
e-mail: s_hashemi@shirazu.ac.ir

A. Dehghantanha (✉)
Department of Computer Science, University of Sheffield, Sheffield, UK
e-mail: A.Dehghantanha@sheffield.ac.uk

A. Dehghantanha et al. (eds.), *Cyber Threat Intelligence*, Advances in Information Security 70, https://doi.org/10.1007/978-3-319-73951-9_7

1 Introduction

Cybercriminals are becoming the main potential threat to all systems connected to the Internet [1]. Malware attacks are increasing while the total number of unique malwares reached to about 600 million in 2016 tripled from 2013 [2]. A botnet is a network of hosts connected to the Internet that is under control of one or more master machine(s) that coordinates each and every bot malicious activities [3, 4]. Botnets have been heavily used to carry different cyber attacks ranging from information espionage, and click-fraud to malware distribution and Distributed Denial of Services (DDoS) [5]. The owner (master) of a botnet can control the botnet using command and control (C&C) software to manage bot clients.

Botnets are implemented in four main topologies namely Star, Multi-Server, Hierarchical and Peer-to-Peer (P2P) [6, 7]. Star topology facilitates a fast and accurate communication between bot nodes in a simple manner but suffers from a single point of failure when spreads commands across the network [7]. Internet Relay Chat (IRC) based botnets are the best example of star C&C [8]. Multi-Server protocol tackles the problem of single point of failure by using several servers as master nodes [7], but its hierarchical structure requires more setup time but allows sharing of the infrastructure. P2P is the most advanced topology of botnets in which every node may serve as a master or client as needed [9]. Although P2P topology has the most latency of convergence but since hides the presence of master nodes it make it more difficult to trace back botnet operation [10]. Due to the potential power and stealthiness of P2P botnets, it is a favorable option among bot masters C&C channels [11, 12]. In this chapter, botnets with one of a few central servers are considered as centralized topology, while Peer-to-Peer botnets is considered as decentralized topology.

While majority of existing research [10, 13–15] are depending on a single topology for botnet detection, this chapter utilizes deep learning to detect botnet traffics independently of underlying botnet architecture. The main contribution of this research is adopting two deep learning techniques namely *Autoencoders* and *Convolutional Neural Networks (CNNs)* to detect malicious botnet traffics. We propose two detection models based on deep learning to eliminate dependency of detection systems to primary features achieved by NetFlow extractor tools. To the best of our knowledge, this is among very first attempts to employ deep learning in botnet detection, therefore this chapter opens new insights in this field. Our models have the capability of detecting malicious traffics from botnets of two common topologies namely centralized and decentralized botnets. It is worth pointing out that *BoTShark* does not pre-filter any primary extracted features and does not need experts' knowledge in selecting proper features to extract features automatically. In other words, *BoTShark* takes samples with all raw features and generates discriminative features for classifying botnets traffics from normal traffics. *BoTShark* is tested on ISCX Botnet Dataset [16] that is a well-known research dataset in the field of botnet detection.

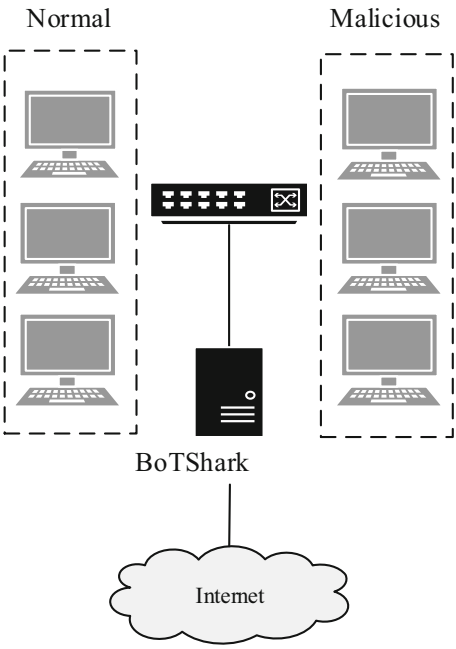

Fig. 1 Malicious and non-malicious systems are connected to the Internet

Figure 1 shows the network architecture considered in this chapter. The network includes both malicious and non-malicious hosts, and *BoTShark* works on the traffics generated by systems connected to the Internet.

The remainder of this chapter is organized as follows. Section 2 reviews some related research. Section 3 gives a brief background on deep Autoencoders and CNNs. Section 4 explains dataset and primary features extracted from network flows. We describe *BoTShark* for detecting botnets traffics in Sect. 5 while Sect. 6 evaluates *BoTShark* models. Finally, Sect. 7 concludes the chapter.

2 Related Work

Botnet is becoming a dominant tool for cybercriminals to distribute new malwares between a large scale of infected host or attacking a target by taking the advantage of cooperative characteristic of the network [17]. The growth in botnet sizes is the main motivation of researches on botnet detection [18]. In the past decade, many researchers attempted to propose a model for detecting bot infected hosts [10, 13, 19] or their malicious network traffics [14, 20, 21]. Most bot host detection systems are relying on profile of each host including their behaviour during a period of time [10] e.g. 6 h [22] or 10 min [23]. BotHunter [24] as one of the earliest botnet behaviour detection system attempts to correlate SNORT [25] generated alarms to behaviour of an individual host. However, it works by inspecting payload of packets that is

not useful against botnets that benefits cryptography in their connections. BotMiner [13] considers group behaviours of individual bots within the same botnet.

On the other hand, some researches focused on detecting malicious botnet traffics originating from internal hosts inside a LAN based on machine learning techniques to predict new NetFlows [14, 15, 20, 26, 27]. Using of machine learning techniques is more common in detecting botnets malicious traffics. A two-phase system consisting of feature extraction phase and machine learning phase is proposed in [20] to detect botnet C&C traffic of IRC based botnets by creating a Bayesian network classifier with 90% detection rate and 15.4% false positive rate that is high. Some works focused on DNS requests [26] and achieved accuracy of 92.5% in detecting malicious DNS requests. However, relying on DNS requests makes the detection system exclusive to botnets that benefits DNS for finding their C&C servers (most centralized botnets). A system is proposed for detecting P2P traffics by assuming that the traffics generated by normal user fluctuate greatly and is different to P2P bots [15]. The system in [15] achieved 98% of detection rate while the false positive rate is still high (30%).

Working on a set of features is a very common approach in the literature of botnet detection while this approach makes the detection systems dependent on the studied network traffics. The features can be suggested by the experts [14, 20, 26, 27] or can be selected by feature selection algorithms [28, 29]. ISOT [30] and ISCX [16] are two well-known datasets in the literature of botnet detection. Zhao et al. [14] attempted to detect malicious traffics by employing a decision tree using the Reduced Error Pruning algorithm (REPTree) and achieved 98.3% true positive rate in detecting malicious traffics, where the model extracts feature vectors for a time window of 60 s and creates a decision tree to classify malicious and non-malicious traffics. However, a research proved that the model proposed in [14] is biased to the dataset [28]. A new stepwise greedy feature selection algorithm for using in botnet traffic detection proposed in [28] that selects best features to differentiate botnet traffics. However, the algorithm works in a greedy manner and it does not consider different permutation of features. Therefore, it is likely to remove valuable feature information. Recently, an approach for detecting botnet traffics based on NetFlows characteristics is proposed [27] that selects best features according to experts analysis on botnets behaviours, where several datasets of botnet traffics are merged to make a dataset for evaluating proposed model. However, combining different datasets requires considering special methods e.g. overlay methods [31] while the details of combining datasets are not described.

Although there are many papers published on detecting bot infected machines or botnet malicious traffics, few models have the capability of detecting malicious traffics independent of botnets topologies. In other words, almost all proposed methods worked on botnets within the same topologies. For example, a model that is proposed to detect decentralized P2P botnets is unable to detect centralized IRC botnet traffics. This chapter aims at differentiating botnet traffics from benign traffics in both topologies.

3 Background: Deep Learning

Deep Learning is a sub field of machine learning that studies artificial neural networks and related machine learning algorithms that contain more than one hidden layer. Since we are using Autoencoders and CNNs, we give a brief background in the following subsections.

3.1 Autoencoders

Artificial Neural Networks (ANN) is one of the popular machine learning techniques that is used for any condition and has the ability of learning problems structure automatically. The learning method in ANN is usually back-propagation in which the samples are forwarded and the output of network is compared with the desired target. Then the error is calculated according to this difference and the weights are tuned [32].

An Autoencoder is similar to a neural network where the output is regarded as the input and supposes that the hidden layer must reconstruct the initial information with the least possible amount of distortion. Then, Autoencoders are trained to reconstruct their own inputs X instead of being trained to predict Y. An Autoencoder tries to learn a function $h_{(W,b)}(x) \cong x$ by learning an approximation to the identity function. As Fig. 2 shows, an Autoencoder always consists of two parts, the encoder and the decoder, which can be a transition between ϕ and ρ such that $\phi : X \rightarrow F$ and $\rho : F \rightarrow X$ where F is the intermediate representation of sample X. An Autoencoder attempts to minimize reconstruction error based on Eq. (1).

$$argmin_{\phi}, \rho \| X - (\rho \circ \phi) X \|^2 \tag{1}$$

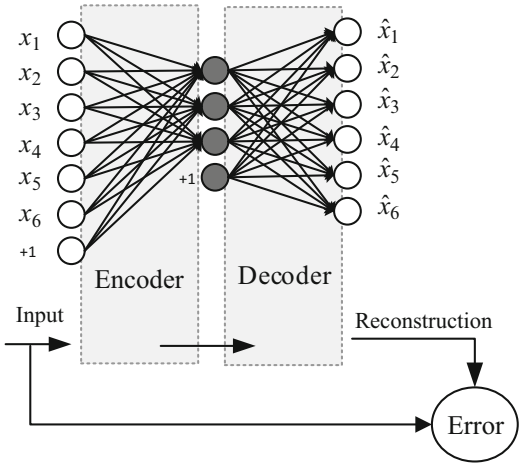

Fig. 2 Stages of an autoencoder that learns the input features in another space

Suppose the dataset only includes a set of unlabeled training examples $X = \{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}\}$, where $x^{(i)} \in R^d$. The aim of an Autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction. Equation (2) shows the hidden layer while Eq. (3) presents the reconstruction from the hidden unit.

$$z = \sigma_1(W^T X + b) \tag{2}$$

$$X' = \sigma_2(Wz + b') \tag{3}$$

The loss function is as Eq. (4) where W and W^T are the transform matrices.

$$L(X, X') = \|X - X'\|^2 = \|X - \sigma_2(W(\sigma_1(W^T X + b)) + b')\|^2 \tag{4}$$

A simple Autoencoder often ends up learning a low-dimensional representation very similar to PCAs and a subspace is learned for the representation of data. If a nonlinear function is exerted in each layer, the nonlinearity is considered and a submanifold is finally extracted that benefits the separability of data points; then in the new space the discrimination procedure is more effective.

3.2 Convolutional Neural Network (CNN)

CNN owes its inception to a well-known research paper [33]. CNNs are usually applied to image but it is also applicable in spam detection, and topic classification, and studies are growing to use CNNs [34].

A CNN consists of different layers that each layer performs a different function on their inputs. Convolution layer, pooling layer and non-linear layer are three layers that are very common in the literature and are usually stacked sequentially many times based on the application [35].

The convolutional layer contains a filter as a sliding window to generate new feature maps. In other words, each filter will generate a new feature set and the model applies different filters to the inputs. The pooling layer uses a kind of filter to summarize the generated feature maps during the previous layer. As convolution explores the correlation between different features, the pooling layer extracts the existing correlation in any parts of the input. There are a few functions namely max pooling, L2-Norm pooling and average pooling that can be used in pooling layer. Since the operations in the layers of convolution and pooling are linear, a non-linear layer such as tanh is used to make the model efficient for non-linear cases.

Assume $X = \{x_1, \ldots, x_n\}$ as the input vector where x_f is value of feature f. If a filter W with M_F weights is applied to the input vector X then Z'_i is the output of convolution layer and is calculated by Eq. (5) for each $i' \in \{1, \ldots, |X| - M_F + 1\}$ where $|X|$ is size of vector X.

$$Z_{i'} = \sum_{i=1}^{M_F} (X_{i'+i-1} W_{M_F-i+1}) \tag{5}$$

Equation (5) considers applying a filter on the input data and there are usually more than one filter with different weights. So Eq. (5) can be applied individually for each filter. Therefore, using of convolution layer on an input vector creates feature maps as its output.

Now the generated feature maps can be fed as the input of max pooling layer to generate reduced feature maps. Using of max pooling is very common in the literature of CNNs [36]. Max pooling outputs y'_i as the maximum value of the features under the swapping filter and is calculated by Eq. (6).

$$y_{i'} = \max_{i \in \Omega(i')} (x_i) \tag{6}$$

where $\Omega(i')$ is the set of feature values starting with i' feature value located under the swapping filter.

Finally, the non-linearity is applied using tanh according to Eq. (7).

$$\tanh x = \frac{1 - e^{2x}}{1 + e^{2x}} \tag{7}$$

4 Data Collection and Primary Feature Extraction

This chapter uses network traffics from ISCX [16] that is created by using one of the most popular overlay methodology introduced in [31] to combine different datasets. The creators of ISCX claims that their dataset has all three requirements of a validated dataset: generality, realism and representativeness. ISCX dataset includes 44.97% of malicious flows from 16 different botnet flows and contains botnet traffics from both centralized and decentralized topologies as well as normal traffics.

This chapter works on network flows extracted from network traffics that might be in form of packet capture (PCAP) files or live traffics. A NetFlow is a set of fields, namely a record, that gives some information about a connection between source and destination (source/destination address, ports etc.). PCAP consists of an Application Programming Interface (API) for capturing network traffic and store them into a file.

This chapter considers all *TCP* and *UDP* NetFlows into a dataset of NetFlows. There are a few tools for extracting NetFlows from network traffics. Argus [37] is a powerful flow exporter that is popular among researchers and is able to extract features from different aspects: byte-based (extracted features based on bytes in a flow e.g. byte sent/received), time-based (features depend on time e.g. inter-packet arrival time) and packet-based features (e.g. total number of packets in a flow).

Table 1 List of features extracted by Argus

Feature	Description
SrcAddr	Source IP address
DstAddr	Destination IP address
Sport	Source port
Dport	Destination port
Proto	Connection protocol (udp, tcp, icmp, igmp etc.)
Dir	Source → Destination, Source ← Destination, Source ↔ Destination
Dur	Record total duration (time between flow start time and flow end time)
TotPkts	Total transaction packet count
SrcPkts	Source → destination packet count
DstPkts	Destination → source packet count
SIntPkt	Source inter-packet arrival time (mSec)
SIntPktIdl	Source idle inter-packet arrival time (mSec)
DIntPkt	Destination inter-packet arrival time (mSec)
DIntPktIdl	Destination idle inter-packet arrival time (mSec)
TotBytes	Total transaction bytes exchanged between source and destination
SrcBytes	Source → destination transaction bytes
DstBytes	Destination → source transaction bytes
sPktSz	Source active interpacket arrival time (mSec)
dPktSz	Destination active interpacket arrival time (mSec)
sMeanPktSz	Mean of the flow packet size transmitted by the source (initiator)
dMeanPktSz	Mean of the flow packet size transmitted by the destination (target)
sMinPktSz	Minimum packet size for traffic transmitted by the source
dMinPktSz	Minimum packet size for traffic transmitted by the destination
sMaxPktSz	Maximum packet size for traffic transmitted by the source
dMaxPktSz	Maximum packet size for traffic transmitted by the destination
Load	Bits per second
SrcLoad	Source bits per second
DstLoad	Destination bits per second
Rate	Packets per second
SrcRate	Source packets per second
DstRate	Destination packets per second

Argus has the capability of extracting 120 features for each connection. However, some features are not useful in the area of botnet detection because they are all null or zero (for example there are several features special to MPLS networks that basically are not relevant to the considered LAN topology). Table 1 shows extracted features by Argus after removing irrelevant features (primary feature selection).

This chapter does not remove any features from the feature set and works on all features of Table 1 (except SrcAddr, DstAddr, Sport and Dport, because the use of these features became inefficient when data comes from different networks [38]). Since Autoencoders and CNNs need continuous features, two categorical features Dir and Proto are converted to continues aspect e.g. 1 and 2 which could handle in

the network. The proposed technique feeds feature vectors for each NetFlow to the learning task. This chapter considers two labels (malicious and non-malicious) for each NetFlow and feeds dataset to *BoTShark*.

5 Proposed BoTShark

We propose *BoTShark* with two deep structures namely *BoTShark-SA* that applies stacked Autoencoders to extract new features for distinguishing malicious and benign network flows and *BoTShark-CNN* which takes the advantage of CNNs to train a classifier for detecting malicious traffics.

5.1 BoTShark-SA: Using Stacked Autoencoders

Figure 3 shows *BoTShark-SA* architecture to learn a classifier that detects malicious NetFlows in a deep structure and to classify new NetFlows as malicious or non-malicious. Since selecting proper features is a big challenge in most machine learning based botnet detection systems [28], *BoTShark-SA* benefits the stacked Autoencoders depicted in Fig. 4. The dataset consists of all unlabeled NetFlows with primary features set $\{f_1, \ldots, f_d\}$ from Sect. 4 is fed to the stacked structure, where d is the dimensionality of data. In the field of deep learning, multiple Autoencoders make an stacked structure to produce the best final denoised output and minimize reconstruction error. This combination learns to extract valuable features in a stepwise manner without employing any particular hand-made feature selection.

Feature Extraction in Fig. 3 extracts efficient features from all given data. As the original number of features are not too much, two layers of Autoencoders are sufficient (see Fig. 4). In the first step of Fig. 4 original feature set is encoded into d_1 features $\{f_1, \ldots, f_{d_1}\}$ to enter the next step. The second Autoencoder receives the output from previous step and extract d_2 features $\{f_1, \ldots, f_{d_2}\}$. The output of layer2 from the second Autoencoder is considered as the final extracted features.

All NetFlows that are now in a new representation, with their corresponding labels provide inputs for training a classifier model to distinguish malicious and non-malicious NetFlows. As the main purpose of deep networks is to extract best features, any classifier can be used in this phase. This chapter input a vector of 27×1 to the first Autoencoder and feed its 20×1 output to the second Autoencoder to extract final ten features (see Fig. 4).

Softmax is a well-known classifier that is using in neural networks [39], it assigns a probability to each class based on the extracted features received from the previous layer to classify the instances. If there are only two classes, *Softmax* acts like a logistic regression which uses a logistic function to assign a probability to instances according to each class. The mentioned function is as Eq. (8).

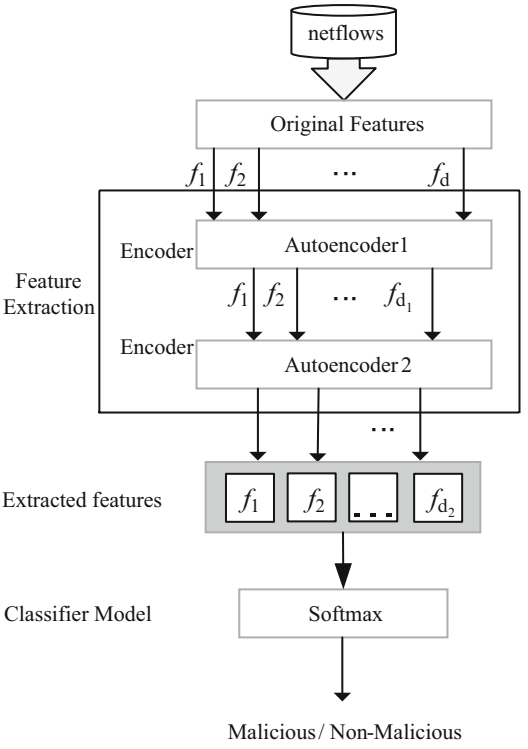

Fig. 3 BoTShark-SA architecture to extract features and train the classifier

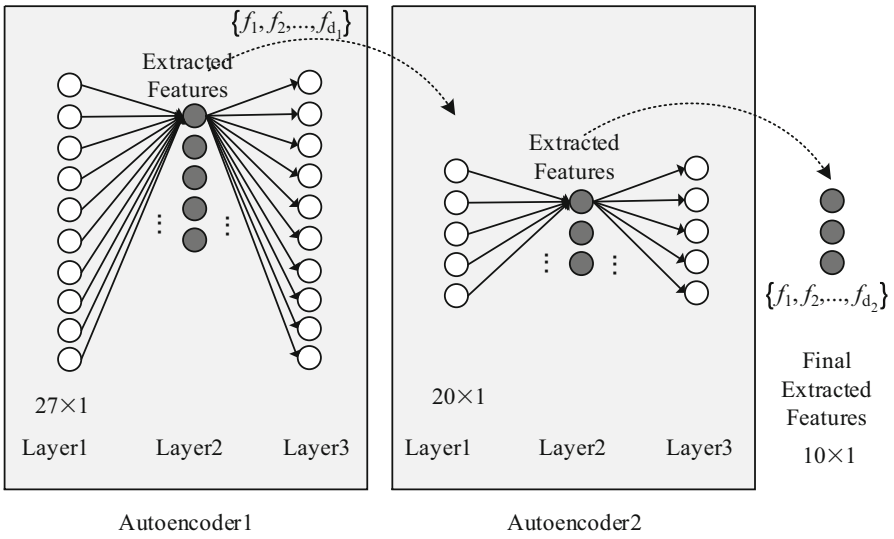

Fig. 4 Stacked Autoencoders that used for extracting features

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}; for\, j = 1, \ldots, K \tag{8}$$

where z is considered as the k-dimensional vector of each sample. Although this chapter is involved with two classes (malicious and non-malicious), *Softmax* can handle multiple classes and the proposed method has the ability to expand to more number of classes by *Softmax*.

5.2 SocialBoTShrak-CNN: Using CNNs

Figure 5 depicts our proposed architecture for training a classifier that has the capability of distinguishing malicious traffics. We input a NetFlow as a vector with 27 extracted features to a CNN for training and feeding calculated weights to a fully connected network with a hidden layer to prepare final inputs for *Softmax* to create final classifier.

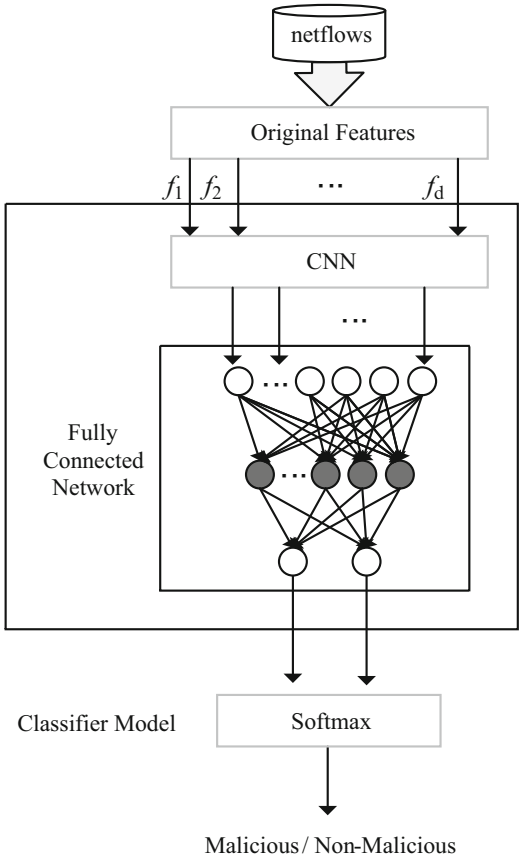

Fig. 5 BoTShark-CNN architecture to train the classifier

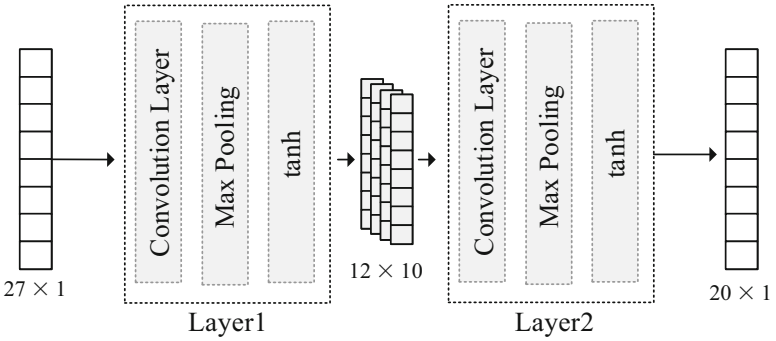

Fig. 6 Implemented architecture for CNNs

We designed a CNN consisting of two layers with three sub-layers of temporal convolution, max pooling and a non-linear function tanh (see Fig. 6). At first all weights are initialized randomly with values between -1 and $+1$ and the structure in Fig. 6 learns appropriate weights automatically for further steps of Fig. 5.

The convolution sub-layer in the first layer of Fig. 6 consists of 10 filters with 4 weights in each filter. This convolution sub-layer turns the 27×1 input vector into 10 feature maps of length 24. These feature maps are then fed to the max pooling sub-layer with filter of width 2 to be converted to 10 feature maps of length 12. After applying tanh sub-layer, we will have feature maps with size of 12×10 given to the second layer. The second layer in its convolution sub-layer applies 5 filters of size 4 that generates 5 feature maps of size 9. Max pooling sub-layer with the filter of size 2 is applied to the feature maps that outputs 5 feature maps of size 4. tanh sub-layer as the final sub-layer is applied to 4×5 feature maps. Finally, these 5 vectors are concatenated to form a 1-D vector of size 20.

We feed our final 20 features to a fully connected neural network with one hidden layer of ten neurons and two outputs in which the neurons apply tanh function. A *Softmax* uses two outputs from the fully connected network to train the final classifier with the capability of detecting malicious network traffics.

6 Evaluation

We are using widely accepted criteria namely True Positive Rate (TPR) and False Positive Rate (FPR) to evaluate our model [40–42]. TPR is reflecting the proportion of positives that are correctly identified and FPR shows the ratio between the number of negative labels wrongly identified as positive. We will also report Receiver Operating Characteristic (ROC) that is a potentially powerful metric for comparison of different classifiers, because it is invariant against skewness of classes in the dataset. In a ROC curve the true positive ratio is plotted in function of the false positive ratio for different thresholds.

We apply *BoTShark* to ISCX botnet dataset to evaluate the performance of the binary classifier in predicting new NetFlows. The proposed technique feeds feature vectors for each NetFlow to the learning task. This chapter considers two labels (malicious and non-malicious) for each NetFlow and feed dataset to *BoTShark*. In this experiment 70% of data is considered as training and the rest 30% for testing the final classifier. *BoTShark-SA* is implemented in Matlab 2015b and *BoTShark-CNN* is implemented using Torch7 framework for deep learning networks and run on a system with 8 GB of RAM and Core i5 of 8 cores of 4 GHz CPU. We set the iteration count of both *BoTShark-SA* and *BoTShark-CNN* to 400 iterations.

Table 2 reflects the performance achieved by lunching *BoTShark* on ISCX dataset. It is demonstrated that *BoTShark-SA* and *BoTShark-CNN* achieved $TPR \geq 0.91$ on ISCX datasets. *BoTShark* works on all primary features and no feature is filtered by experts. *BoTShark-SA* achieved TPR of 0.91 and its false positive ratio is 0.15 and *BoTShark-CNN* achieved higher detection ratio (0.92).

Figure 7 depicts the ROC diagrams of *BoTShark-SA* and *BoTShark-CNN*. The diagram shows True Positive Rate (X-Axis) against False Positive Rate (Y-Axis) to demonstrate the amount of false positives for achieving a specified true positives. The ROC diagrams of *BoTShark* shows that in false positive ratio ≤ 0.05 we will have true positive ratio of about 0.75. However, *BoTShark* achieves higher true positive ratios ($\geq$0.91) by tolerating more false positives (FPR between 0.05 and 0.15).

Table 2 Performance of BoTShark in detecting malicious traffics

	TPR	FPR
BoTShark-SA	0.91	0.13
BoTShark-CNN	0.92	0.15

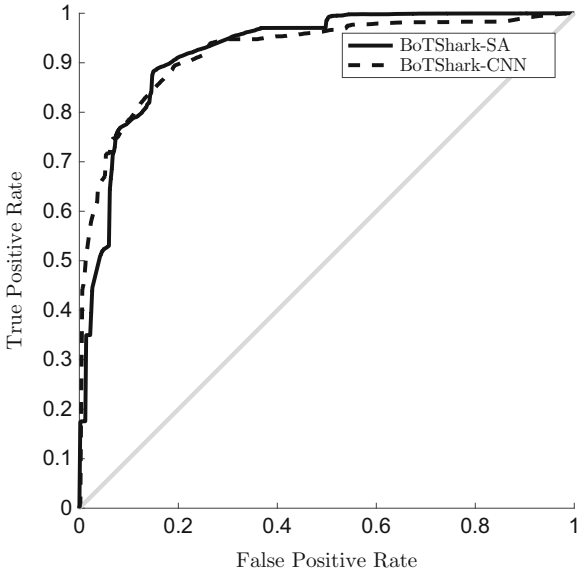

Fig. 7 ROC diagrams of BoTShark-SA and BoTShark-CNN

Working directly on the outputs from Argus and extracting features automatically without experts is an advantage of *BoTShark*. In other words, the main advantage of the proposed method is the variety of applications. Since Autoencoders and CNNs are not limited to a special field of data, it can be used for efficient feature extraction in any cases. Then any NetFlow extractor such as Argus [37], YAF [43] and ISCXFlowMeter [44] can be used as the flow exporter.

7 Conclusion

IRC and P2P are two main botnet topologies hence a typical network may include infected hosts from both topologies and new detection systems are required to support both. As deep learning is very efficient in image processing and text mining, this chapter attempted to apply deep learning techniques in the realm of botnet detection by proposing *BoTShark-SA* that uses Autoencoders and *BoTShark-CNN* which uses CNN. *BoTShark* has the ability of detecting botnet traffics from both common topologies of botnets namely centralized and P2P. A *Softmax* classification makes the final predictor of malicious and non-malicious traffics. We achieved *TPR* of 0.91 with FPR of 0.13 in detecting malicious traffics of botnets. Our study also showed that Autoencoders perform better than CNN since as it generates smaller false positives. Applying other deep learning techniques such as Long Short Term Memory (LSTM) can be considered as a future work of this study. Moreover, the approach of this study can be applied for detection of relevant evidences during course of forensics investigation of cloud [45] and IoT [46] botnet traffic as well.

References

1. Nikola Milosevic, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Machine learning aided android malware classification. *Computers & Electrical Engineering*, feb 2017. https://doi.org/10.1016/j.compeleceng.2017.02.013.
2. Malware statistics & trends report, feb 2017. https://www.av-test.org/en/statistics/malware/.
3. Mohsen Damshenas, Ali Dehghantanha, and Ramlan Mahmoud. A survey on malware propagation, analysis, and detection. *International Journal of Cyber-Security and Digital Forensics (IJCSDF)*, 2(4):10–29, 2013.
4. Mohsen Damshenas, Ali Dehghantanha, Kim-Kwang Raymond Choo, and Ramlan Mahmud. M0droid: An android behavioral-based malware detection model. *Journal of Information Privacy and Security*, 11(3):141–157, jul 2015. https://doi.org/10.1080/15536548.2015.1073510.
5. Hamed Haddad Pajouh, Reza Javidan, Raouf Khayami, Dehghantanha Ali, and Kim-Kwang Raymond Choo. A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in IoT backbone networks. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, 2016. https://doi.org/10.1109/tetc.2016.2633228.
6. Gunter Ollmann. Botnet communication topologies understanding the intricacies of botnet command-and-control, 2009. http://technicalinfo.net/papers/PDF/WP_Botnet_Communications_Primer_(2009-06-04).pdf.

7. Anoop Chowdary Atluri and Vinh Tran. Botnets threat analysis and detection. In *Information Security Practices*, pages 7–28. Springer International Publishing, 2017. https://doi.org/10.1007/978-3-319-48947-6_2.
8. Hossein Rouhani Zeidanloo, Mohammad Jorjor Zadeh Shooshtari, Payam Vahdani Amoli, M. Safari, and Mazdak Zamani. A taxonomy of botnet detection techniques. In *2010 3rd International Conference on Computer Science and Information Technology*. IEEE, jul 2010. https://doi.org/10.1109/iccsit.2010.5563555.
9. C.C. Zou and R. Cunningham. Honeypot-aware advanced botnet construction and maintenance. In *International Conference on Dependable Systems and Networks (DSN06)*. IEEE, 2006. https://doi.org/10.1109/dsn.2006.38.
10. Junjie Zhang, Roberto Perdisci, Wenke Lee, Xiapu Luo, and Unum Sarfraz. Building a scalable system for stealthy p2p-botnet detection. *IEEE Transactions on Information Forensics and Security*, 9(1):27–38, jan 2014. https://doi.org/10.1109/tifs.2013.2290197.
11. Yee-Yang Teing, Ali Dehghantanha, Kim-Kwang Raymond Choo, and Laurence T Yang. Forensic investigation of p2p cloud storage services and backbone for IoT networks: BitTorrent sync as a case study. *Computers & Electrical Engineering*, 58:350–363, feb 2017. https://doi.org/10.1016/j.compeleceng.2016.08.020.
12. Opeyemi Osanaiye, Haibin Cai, Kim-Kwang Raymond Choo, Ali Dehghantanha, Zheng Xu, and Mqhele Dlodlo. Ensemble-based multi-filter feature selection method for DDoS detection in cloud computing. *EURASIP Journal on Wireless Communications and Networking*, 2016 (1), may 2016. https://doi.org/10.1186/s13638-016-0623-3.
13. Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 139–154, Berkeley, CA, USA, 2008. USENIX Association. http://dl.acm.org/citation.cfm?id=1496711.1496721.
14. David Zhao, Issa Traore, Bassam Sayed, Wei Lu, Sherif Saad, Ali Ghorbani, and Dan Garant. Botnet detection based on traffic behavior analysis and flow intervals. *Computers & Security*, 39:2–16, nov 2013. https://doi.org/10.1016/j.cose.2013.04.007.
15. Basil AsSadhan, Jose M. F. Moura, and David Lapsley. Periodic behavior in botnet command and control channels traffic. In *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference*. IEEE, nov 2009. https://doi.org/10.1109/glocom.2009.5426172.
16. Unb iscx botnet dataset, jan 2017. http://www.unb.ca/research/iscx/dataset/ISCX-botnet-dataset.html#Botnet%20Data%20set.
17. Sérgio S.C. Silva, Rodrigo M.P. Silva, Raquel C.G. Pinto, and Ronaldo M. Salles. Botnets: A survey. *Computer Networks*, 57(2):378–403, feb 2013. https://doi.org/10.1016/j.comnet.2012.07.021.
18. Igal Zeifman. 2015 bot traffic report: Humans take back the web, bad bots not giving any ground. Report, Incapsula, 9 Dec. 2015 2015. https://www.incapsula.com/blog/bot-traffic-report-2015.html.
19. Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security - CCS*. ACM Press, 2009. https://doi.org/10.1145/1653662.1653738.
20. Carl Livadas, Robert Walsh, David Lapsley, and W. Strayer. Usilng machine learning technliques to identify botnet traffic. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*. IEEE, nov 2006. https://doi.org/10.1109/lcn.2006.322210.
21. Chia Yuan Cho, Domagoj Babic, Eui Chul Richard Shin, and Dawn Song. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM conference on Computer and communications security - CCS*. ACM Press, 2010. https://doi.org/10.1145/1866307.1866355.
22. Rouhani S. Zeidanloo HR. *Botnet Detection by Monitoring Common Network Behaviors: Botnet detection by monitoring common network behaviors*. Lambert Academic Publishing, 2012. ISBN 9783848404759.

23. Jing Wang and Ioannis Ch. Paschalidis. Botnet detection based on anomaly and community detection. *IEEE Transactions on Control of Network Systems*, pages 1–1, 2016. https://doi.org/10.1109/tcns.2016.2532804.
24. Guofei Gu, Phillip Porras, Vinod Yegneswaran, and Martin Fong. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *16th USENIX Security Symposium (USENIX Security 07)*, Boston, MA, 2007. USENIX Association. https://www.usenix.org/conference/16th-usenix-security-symposium/bothunter-detecting-malware-infection-through-ids-driven.
25. Snort - network intrusion detection & prevention system, jan 2017. https://www.snort.org/.
26. Hyunsang Choi, Hanwoo Lee, Heejo Lee, and Hyogon Kim. Botnet detection by monitoring group activities in DNS traffic. In *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*. IEEE, oct 2007. https://doi.org/10.1109/cit.2007.90.
27. G. Kirubavathi and R. Anitha. Botnet detection via mining of traffic flow characteristics. *Computers & Electrical Engineering*, 50:91–101, feb 2016. https://doi.org/10.1016/j.compeleceng.2016.01.012.
28. Elaheh Biglar Beigi, Hossein Hadian Jazi, Natalia Stakhanova, and Ali A. Ghorbani. Towards effective feature selection in machine learning-based botnet detection approaches. In *2014 IEEE Conference on Communications and Network Security*. IEEE, oct 2014. https://doi.org/10.1109/cns.2014.6997492.
29. Kuan-Cheng Lin, Wei-Chiang Li, and Jason C. Hung. Detection for different type botnets using feature subset selection. In *Lecture Notes in Electrical Engineering*, pages 523–529. Springer Singapore, 2016. https://doi.org/10.1007/978-981-10-0539-8_52.
30. Isot botnet dataset, jan 2017. http://www.uvic.ca/engineering/ece/isot/datasets/.
31. Adam J. Aviv and Andreas Haeberlen. Challenges in experimenting with botnet detection systems. In *Proceedings of the 4th Conference on Cyber Security Experimentation and Test*, CSET'11, pages 6–6, Berkeley, CA, USA, 2011. USENIX Association. http://dl.acm.org/citation.cfm?id=2027999.2028005.
32. Jeff Heaton. *Artificial Intelligence for Humans*, volume 3. CreateSpace Independent Publishing Platform, 2015. ISBN 1505714346.
33. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. https://doi.org/10.1109/5.726791.
34. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.
35. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, may 2015. https://doi.org/10.1038/nature14539.
36. Michael Blot, Matthieu Cord, and Nicolas Thome. Max-min convolutional neural networks for image classification. In *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, sep 2016. https://doi.org/10.1109/icip.2016.7533046.
37. Argus- auditing network activity, jan 2017. http://qosient.com/argus.
38. Huy Hang, Xuetao Wei, M. Faloutsos, and T. Eliassi-Rad. Entelecheia: Detecting p2p botnets in their waiting stage. In *2013 IFIP Networking Conference*, pages 1–9, May 2013.
39. Bishop Christopher. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 1 edition, 2006.
40. Sajad Homayoun, Ali Dehghantanha, Marzieh Ahmadzadeh, Sattar Hashemi, and Raouf Khayami. Know abnormal, find evil: Frequent pattern mining for ransomware threat hunting and intelligence. *IEEE Transactions on Emerging Topics in Computing*, pages 1–1, 2017 - In Press. https://doi.org/10.1109/tetc.2017.2756908.
41. Mina Sohrabi, Mohammad M. Javidi, and Sattar Hashemi. Detecting intrusion transactions in database systems: a novel approach. *Journal of Intelligent Information Systems*, 42(3):619–644, dec 2013. https://doi.org/10.1007%2Fs10844-013-0286-z.

42. R. Mohammadi, R. Javidan, and M. Conti. Slicots: An sdn-based lightweight countermeasure for tcp syn flooding attacks. *IEEE Transactions on Network and Service Management*, 14(2): 487–497, June 2017. ISSN 1932-4537. https://doi.org/10.1109/TNSM.2017.2701549.
43. Yet another flowmeter, jan 2017. https://tools.netsa.cert.org/yaf/.
44. Gerard Drapper Gil, Arash Habibi Lashkari, Mohammad Mamun, and Ali A. Ghorbani. Characterization of encrypted and vpn traffic using time-related features. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy ICISSP 2016*, pages 407–414, 2016.
45. Yee-Yang Teing, Ali Dehghantanha, Kim-Kwang Raymond Choo, Tooska Dargahi, and Mauro Conti. Forensic investigation of cooperative storage cloud service: Symform as a case study. *Journal of Forensic Sciences*, 62(3):641–654, nov 2016. https://doi.org/10.1111/1556-4029.13271.
46. Steve Watson and Ali Dehghantanha. Digital forensics: the missing piece of the internet of things promise. *Computer Fraud & Security*, 2016(6):5–8, jun 2016. https://doi.org/10.1016/s1361-3723(15)30045-2.