DICCh-D: Detecting IPv6-Based Covert Channels Using DNN

Arti Dua[1](✉), Vinita Jindal[2], and Punam Bedi[3]

[1] Bhaskaracharya College of Applied Sciences, University of Delhi, Delhi, India
arti.batra@bcas.du.ac.in

[2] Keshav Mahavidyalaya, University of Delhi, Delhi, India
vjindal@keshav.du.ac.in

[3] Department of Computer Science, University of Delhi, Delhi, India
pbedi@cs.du.ac.in

Abstract. Stegomalware uses Information hiding techniques within Network protocols to leak out sensitive data and/or to exchange hidden commands between secretly communicating parties. Internet Protocol version 6 (IPv6) is a network layer protocol that is rapidly replacing Internet Protocol version 4 (IPv4). This has resulted in an increase in the availability of IPv6 packets over the internet, thereby making IPv6 a good candidate for the establishment of Network covert channels (NCCs). NCCs use either network flows or network packets to communicate in a hidden way such that no one notices the presence of secret information inside them. This secret information can be injected into network flows or network packets in the inter-packet delays and/or in any of the redundant, unused, reserved storage areas of the packets. Substantial research work has already been done in the area of covert channel detection in IPv4. Now, with IPv6 taking over the internet, the focus has shifted towards the detection of possible covert channels in IPv6. Thus this paper proposes DICCh-D, a model for the Detection of IPv6-based Covert Channels using DNN. For experimentation, the dataset was constructed using normal IPv6 packets obtained from CAIDA's dataset and covert IPv6 packets created by running an IPv6-based covert packets generation tool. The proposed method outperforms CNN, LSTM and SVM in terms of accuracy with acceptable training and testing time. DICCh-D attained an accuracy of 99.59% and a recall value of 0.99, which is better than the existing technique used in literature.

Keywords: Network covert channel detection · Stegomalware · IPv6 · CAIDA dataset · Deep neural network

1 Introduction

With the growth of technology, the attackers have started using sophisticated and new techniques to perform various attacks like crypto lockers, advanced persistent threats (APT), etc. Stegomalware concerns the transfer of malware through some form of Information hiding and is being used by attackers extensively [1]. Stegomalware offers hidden transfer of malware in legitimate Internet traffic flows offering reasonable undetectability. Network covert channels (NCCs) utilize network protocols for Information Hiding that can further be used as a medium for performing above mentioned attacks. NCCs can be categorized into three types:

1. Storage-Based Network Covert Channels: These NCCs utilize the storage area of the header or payload part of a protocol to hide secret information.
2. Timing-Based Network Covert Channels: These types of NCCs utilize the inter-packet delays between consecutive packets to hide secret information.
3. Hybrid Network Covert Channels: These types of NCCs utilize both storage and timing-based Network covert channels simultaneously.

The efficiency of any NCC depends upon three characteristics:

1. Information hiding capacity of the covert channel
2. The undetectability of the hidden information
3. The robustness of the hiding algorithm used to develop NCC.

The usability of the Network protocol used in the NCC also affects the Information hiding capacity of the NCC. The Network protocols that have higher usability over the networks offer higher covert capacity. With IPv6 rapidly replacing IPv4 over the Internet, the usability of IPv6 packets over the internet is also increasing day by day. This makes IPv6 a good target for hidden communication. Some of the possible storage-based network covert channels that may be developed using the IPv6 protocol have been proposed by researchers in [2, 3]. A lot of work has already been done on the detection of IPv4-based covert channels. To the best of our knowledge, the area of IPv6-based covert channel detection is comparatively less explored. Thus this paper proposes DICCh-D, a model to detect IPv6-based covert channels using Deep Neural Network. For experimentation, the dataset was created using normal IPv6 packets obtained from CAIDA's dataset of Anonymized Internet Traces 2019 [4] and covert IPv6 packets generated by running the pcapStego tool [5].

The further organization of this paper is as follows. Section 2 briefly discusses the IPv6 Protocol and the Deep Neural Network. In Sect. 3, the related research work done in the area of detection of IPv6-based NCCs is discussed. Section 4 elaborates on the details and working of the proposed DICCh-D. Section 5 discusses the construction of the dataset, experiments, and results followed by Sect. 6 which concludes this paper.

2 Background Information

This section gives a brief description of IPv6 Protocol and the Deep Neural Network algorithm which are used in this paper.

2.1 Internet Protocol Version 6

The Internet Engineering Task Force (IETF) developed version 6 of the Internet Protocol in the year 1998 to overcome the problem of exhausting 32 bits long IPv4 addresses.

RFC 8200 [6] provides a detailed description of the IPv6 protocol. It is soon expected to take over its prevalent predecessor IPv4 and is termed the next-generation protocol. According to Google statistics, the adoption rate for IPv6 reached a value of 37.25% on January 15, 2022, which is continuously increasing [7]. The header structure of this protocol is depicted in Fig. 1 below.

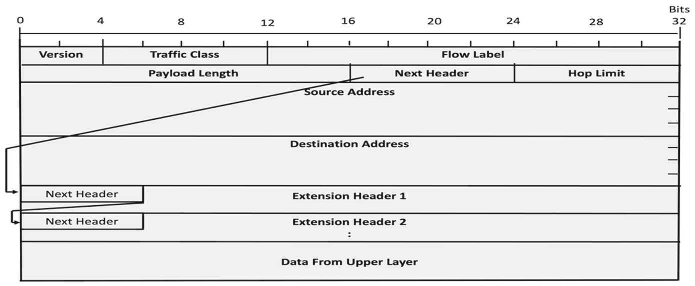

Fig. 1. Header structure of IPv6 protocol

The first field in the header is the Version field. It is 4 bits long and contains the version number of the Internet Protocol being used. For IPv6 packets, the value of this field is fixed to 6. The second field in the IPv6 header is the Traffic Class (TC) field. This field is 1 byte long and is used for Network Traffic Management. The subsequent field in the IPv6 header is the Flow Label (FL) field whose size is 20 bits long and is used to identify the packets belonging to a single flow. The fourth field in the sequence is the Payload Length (PL) field which is 2 bytes long in size and stores the length of both the extension headers (if any) and the data from the upper layer protocols. Next comes, a 1-byte long Next Header (NH) field that stores the protocol number of the extension header attached next to the fixed IPv6 header. Extension headers (EH) like Hop by Hop EH, Destination EH, Routing EH, Fragmentation EH, Authentication EH, and Encrypted Payload Security EH are some extension headers specified in RFC 8200 [6]. The common upper layer extension headers are the Transmission Control Protocol header and User Datagram Protocol header. The next field in the IPv6 header is a 1 byte long Hop Limit (HL) field that defines the number of nodes that an IPv6 packet can traverse without getting discarded over the Internet. Next to it are Source and Destination Address Field. Both of these fields are 128 bits long and hold the logical source and destination address of an IPv6 packet. Amongst all the header fields of IPv6, the highest storage-based covert capacity is offered by the Flow Label field which is 20 bits. Further, the Traffic class field also offers a good hiding capacity of 8 bits per IPv6 packet. The randomness in the legitimate values of these two fields makes them suitable candidates for covert channel creation.

2.2 DNN Model

A DNN is a machine learning algorithm. It is a special type of neural network with three types of layers: Input Layer L_i (One in number), Hidden Layers $L_{H1}, L_{H2}, \dots L_{HN}$

(two or more in number), and Output Layer L_o (One in number). Every layer contains a certain number of neurons. Further, a neuron at one layer is connected to every other neuron at the next layer in the forward direction making it a feed-forward network and each of these connections is assigned some weight W_{xy}. There is no connection between neurons at the same layer. During the training phase of a model, the algorithm tries to find the best weight for each connection through back-propagation. The training of a DNN starts with the input layer. Each neuron at this layer receives input and multiplies this received value with the initial weights assigned to its connection with neurons at the next layer. Further, these product values are forwarded to the hidden layer, L_{H1} where a non-linear activation function is applied to the weighted input value received at each neuron, from the previous layer. In this way, each subsequent hidden layer extracts more significant information. The hidden layer L_{HN} processes the information and forwards it finally to the output layer L_o which gives a probability of the current input belonging to the available classes/labels. If the class with the highest probability value for this input does not match the corresponding actual class, the output value is sent back to the initial layers to adjust the errors and update the weights again.

After the completion of the training phase, the final weights of the DNN are used to make predictions on the testing data to test the generalization capability of the trained model.

3 Related Work

The idea of the development of covert channels over Network communication protocol was first given by Handel and Sandford in the year 1996 [8]. In this work, the authors discussed how various network protocols operating at various layers of the OSI model can be exploited for developing covert channels for secret and hidden communications. Over the years, various researchers also proposed several techniques for the development of covert channels over different network protocols used over the LANs (Local Area Network) and the Internet including IPv4, TCP, UDP, ARP, ICMP [9–14]. Comparatively, IPv6 being a younger protocol was explored by Lucena et al. [2] in the year 2005 for the possibility of the development of covert channels. The authors suggested the possibility of 22 different storage-based covert channels using different parts of the IPv6 header. In 2019, Mazurczyk et al. practically experimented with the possibility and actual bandwidth of covert channels proposed by Lucena et al. over the internet [3]. The authors used the following six IPv6 header fields individually to develop covert channels namely Traffic Class, Flow Label, Payload Length, Next Header, Hop Limit, and Source Address field. Bedi et al. proposed an IPv6-based covert channel that utilized the presence/absence of an extension header in a fixed predefined order to covertly convey a 0/1 bit at respective positions [15].

Further, this paper discusses the work done in the detection of IPv6-based Network covert channels. In 2006, Lewandowski et al. suggested the elimination of covert channels based on Routing extension headers and Hop Limit using Traffic normalization with the help of active wardens in IPv6 networks [16]. In [17], Luca et al. suggested the use of code augmentation in extended Berkeley Packet Filter (eBPF) within Linux kernel to collect the statistics of IPv6 header fields. Repetto et al. suggested the use of the BCC tool for running eBPF programs to obtain statistics about three specific header fields in IPv6 viz. Flow Label, Traffic Class, and Hop Limit [18]. The authors inferred that abnormal changes in the statistical values of these header fields can raise an alarm about the presence of a covert channel. But this eBPF-based covert channel detection mechanism gives good accuracy with more granular values of the number of bins which consumes a large amount of resources. Also, short covert communications cannot be detected using the same mechanism. Salih et al. used a Naive Bayes classifier which is a Machine Learning technique to detect IPv6-based covert channels with an accuracy of 94.47% [19]. They proposed a framework that uses a hybrid method for feature selection that uses Intelligent Heuristic Algorithm (IHA) in addition to a modified Decision Tree C4.5 to create primary training data to detect hidden channels in the IPv6 network. AI Senaid in his work [20] applied a CNN-based approach to identify covert channel created in the code field of ICMPv6 protocol. To the best of our knowledge, there is a scope for improvement in prediction accuracy or resource consumption for the detection of IPv6-based covert channels. Thus, in this paper, DICCh-D, a model to detect IPv6-based covert channels using DNN is proposed. The next section describes the proposed detection model.

4 The Proposed DICCh-D

This paper proposes DICCh-D, a model that detects IPv6-based covert channels using DNN. The development of DICCh-D is divided into three phases as shown in Fig. 2. The first phase creates the IPv6 packets dataset containing both covert and normal IPv6 packets. The second phase extracts the header fields of these IPv6 packets to create a new dataset of header fields of covert and normal IPv6 packets. In the third phase, the training and validation datasets are used to train and validate the DNN whereas the testing dataset is used to test the generalization capability of the trained model. Each of the phases is explained further in the sub-sections.

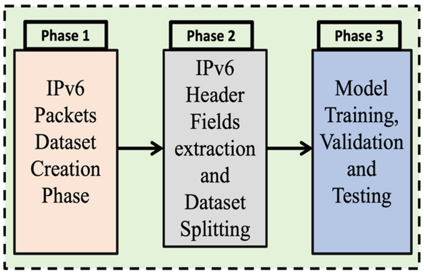
Phase 1
IPv6 Packets Dataset Creation Phase
Phase 2
IPv6 Header Fields extraction and Dataset Splitting
Phase 3
Model Training, Validation and Testing

Fig. 2. Development framework of proposed DICCh-D

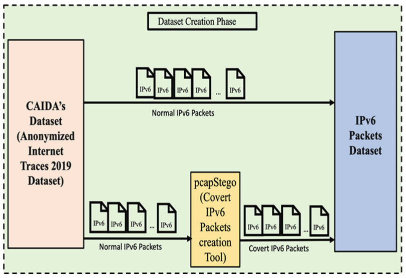
Dataset Creation Phase
CAIDA's Dataset (Anonymized Internet Traces 2019 Dataset)
Normal IPv6 Packets
Normal IPv6 Packets
pcapStego (Covert IPv6 Packets creation Tool)
Covert IPv6 Packets
IPv6 Packets Dataset

Fig. 3. IPv6 packets dataset creation phase

4.1 Phase 1: IPv6 Packets Dataset Creation

The IPv6 packet dataset consisted of normal IPv6 packets and covert IPv6 packets. The process of dataset creation is shown in Fig. 3. The normal IPv6 packets were obtained randomly from CAIDA's Anonymized Internet Traces 2019 [4].

The covert IPv6 packets were created with the help of a tool called pcapStego [5]. This tool offers the creation of storage-based covert packets in IPv6 header fields. After the creation of the IPv6 packets dataset containing normal IPv6 packets and covert IPv6 packets, the next step was the extraction of IPv6 header fields to create a new dataset.

4.2 Phase 2: Extraction of IPv6 Header Fields from IPv6 Packets Dataset

A Network protocol packet consists of two vital components, a packet header, and a payload. A packet header contains the metadata about the current packet and the payload part contains the actual information carried by the packet over a network. In this paper, only the detection of storage-based network covert channels over IPv6 is considered. Normal IPv6 packets were obtained from Anonymized Internet Traces 2019 obtained from CAIDA [4]. The covert IPv6 packets were created with the help of a tool named pcapStego [5]. The .pcap files for both normal and covert data packets were obtained and the following header fields were fetched from these packets: Flow Label, Traffic Class, Payload Length, Hop Limit, Next Header, Source IPv6 address, Destination IPv6 Address, Source Port, Destination Port, Transport Layer Protocol. The task of extraction of IPv6 header fields was done with the help of the Wireshark tool [21]. After this, the headers of normal IPv6 packets and headers of covert IPv6 were combined together to form a complete dataset with labels. Subsequently, the complete dataset was divided into the training_and_validate dataset (80% of the complete dataset) and the testing dataset (20% of the complete dataset).

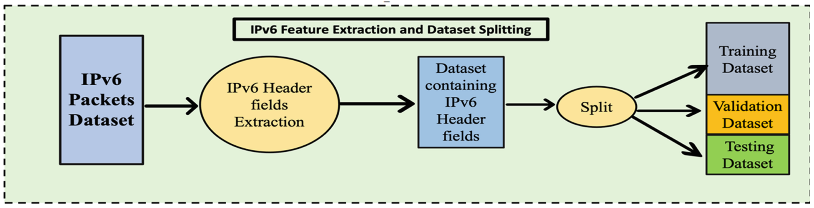
IPv6 Feature Extraction and Dataset Splitting
IPv6 Packets Dataset
IPv6 Header fields Extraction
Dataset containing IPv6 Header fields
Split
Training Dataset
Validation Dataset
Testing Dataset

Fig. 4. IPv6 header fields extraction

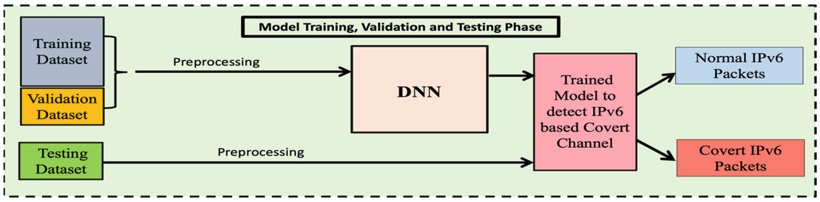

Fig. 5. Model training, validation and testing phase

The training_and_validate dataset was further split into the training dataset (90% of the training_and_validate dataset) and validation dataset (10% of the training_and_validate dataset) to check the performance of the DNN after each epoch on the validation dataset while training the DNN. The complete working of this phase is shown in Fig. 4.

4.3 Phase 3: Dataset Preprocessing, Training and Testing of the DNN

In this phase, three subtasks of dataset preprocessing, training, and testing the DNN are performed as shown in Fig. 5. Before the training phase, the training dataset and the validation dataset are preprocessed to quantize and standardize the data. This preprocessed training dataset is used to train the DNN. The structure of the DNN consisted of one input layer, three hidden layers (with 24, 12, and 6 neurons), and one output layer. The activation function used at the hidden layers was Rectified Linear Unit (ReLU). The activation function used at the output layer was sigmoid. The values fed to the output layer of the DNN are converted to respective values between 0 and 1 with the help of the sigmoid activation function to make the predictions.

5 Experimental Study

The proposed DICCh-D was developed using a 1.8 GHz Dual-Core Intel Core i5 processor on macOS Catalina. Python version 3.6.8 was used for the development of the proposed DICCh-D. The development of DICCh-D consisted of the creation of a dataset, preprocessing of the dataset, training of the DNN, and testing of the DNN.

5.1 Dataset

The dataset for training and testing of the DICCh-D consisted of normal IPv6 packets obtained from CAIDA's dataset (Anonymized Internet Traces 2019) and covert IPv6 packets obtained using the pcapStego tool. The pcapStego tool hides covert data in Flow Label, Traffic Class, and Hop limit fields individually for each IPv6 packet. Since Flow label (20 bits) and Traffic class (8 bits) offer good hiding capacity for covert communications, the covert data packets were generated using Traffic Class or Flow Label fields randomly. The pcapStego tool needs to input a .pcap file that contains normal IPv6 packets which can be used to hide data. This .pcap file was also obtained from CAIDA's Anonymized Internet Traces IPv6 2019 dataset. The final combined IPv6 packets dataset was then created by combining the normal IPv6 packets and IPv6 packets carrying covert data. The combined IPv6 packets dataset contained 16091 IPv6 packets in all. Out of a total of 16091 packets, 9460 were normal IPv6 packets, and the rest 6631 were covert IPv6 packets. Further, Wireshark software was used to fetch the relevant header fields of captured IPv6 packets and convert that to a.csv file. This.csv file consisted of the following header field attributes: Flow Label, Traffic Class, Payload Length, Hop Limit, Next Header, Source IPv6 address, Destination IPv6 Address, Source Port, Destination Port, and Transport Layer Protocol.

Testing the generalizability of a trained model is an important aspect in assuring how a model will perform on the data it has never seen earlier. For this, the complete dataset of 16091 packets was divided into two parts, the training_and_validation dataset, and the testing dataset. The training_and_validation dataset consisted of 12872 packets and the testing dataset consisted of 3219 packets. The training_and_validation dataset contained 7584 normal IPv6 packets and 5288 covert IPv6 packets. The testing dataset contained 1876 normal packets and 1343 covert IPv6 packets. The first dataset was used to train and validate the model. The second dataset was used to test the generalization ability of the model trained with the first dataset. The preprocessing of both of these datasets was done independently using the same python program to quantize and standardize the values. Section 5.2 describes the preprocessing applied to both datasets separately.

5.2 Preprocessing

In this work, the following steps were done using a python program to preprocess the datasets containing normal and covert IPv6 packets. Firstly, a single attribute corresponding to the Source IPv6 Address field having colon-separated 8 octets was broken into 8 different attributes. Similarly, the Destination IPv6 address was broken down into 8 different attributes. The dataset contained two categorical attributes: Protocol and Next Header. Quantization was used to convert these categorical values into a unique number corresponding to different values of each attribute. Input attributes have different scales and hence there is a need for scaling or standardization in ML algorithms. In this work, *standardscalar()* of *sklearn* library from python was used to scale all the data. The next sub-section describes the training and testing Phase of the proposed DICCh-D.

5.3 Training and Testing Phase

Before starting with the training phase, the preprocessing of the training_and_validation dataset and the testing dataset was done separately. At the beginning of the training phase, the preprocessed training_and_validation dataset was divided into two parts: the training dataset (used solely for the purpose of training the DNN) and the validation dataset (to check the performance of the model after each epoch during the training phase). During the training phase of any Machine learning/Deep Learning algorithm, certain hyperparameters values need to be set such as the number of layers used for training a model, the number of neurons used in each layer of the model, the batch size, the learning rate, the number of epochs needed for training, the optimizer, and the activation function used at each layer. These hyperparameters have to be chosen carefully as it has a significant effect on the performance of the model. Hence the DNN was trained with different configurations like 2, 3, 4 number of hidden layers, the batch size of 10, 32, and the number of epochs from 30 to 50 with an interval of 10. The most optimal hyper-parameters were then decided as the final configuration as follows: number of hidden layers as 3, number of neurons at three hidden layers as 24, 12, and 6 respectively, and the activation function used at the hidden layers was ReLU. At the output layer, the sigmoid activation function was used. The batch size was fixed at 10 and the number of epochs was fixed at 30. The optimizer used was AdamOptimizer. The evaluation metrics used for the evaluation of the DICCh-D are described in Subsect. 5.4.

5.4 Evaluation Metrics

To measure the effectiveness of the proposed DICCh-D, metrics like accuracy, precision, recall, and F1-score were calculated for the testing dataset that was used to check the generalizability of the proposed model. True Positives (TP) is the number of samples classified correctly by a model. False Positives (FP) depicts the number of samples mistakenly classified as being positive by a model. True Negatives (TN) denotes the number of samples which are correctly classified by a model as being negative. False Negatives (FN) depicts the number of samples which are incorrectly classified as being negative. Accuracy describes the total number of samples which are correctly classified by the model. Precision denotes the number of samples actually belonging to a class out of all the samples that were predicted by a model as belonging to that class. Recall is defined as the number of samples correctly predicted by a model out of all the samples belonging to that class. F1-Score is defined as the harmonic mean of the Recall and Precision value. Equations (1) to (4) are used to calculate the Accuracy, Precision, Recall, and F1-Score of a model.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 - Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4}$$

The evaluation results of all experiments done using the above-mentioned metrics for different configurations of models in consideration are discussed in the next sub-section.

5.5 Results

The performance of DICCh-D was compared with state-of-the-art Deep-Learning algorithms (CNN, LSTM) and a Machine Learning algorithm (SVM). The CNN was experimented with three different configurations of 1, 2, and 3 1D-Convolutional layer(s), each with a batch size of 10, adopting AdamOptimizer as the optimizer and iterated over 50 epochs. Next, the LSTM model was iterated over 25 epochs each time for three configurations of 1, 2, and 3 LSTM layer(s). Finally, the SVM model was experimented with four different kernel values viz. Linear, sigmoid, polynomial, and RBF. The proposed DICCh-D was experimented with three different configurations of DNN each having 2, 3, and 4 hidden layers respectively. The results for accuracy for each of the classifiers taking different hyperparameters are shown in Fig. 6.

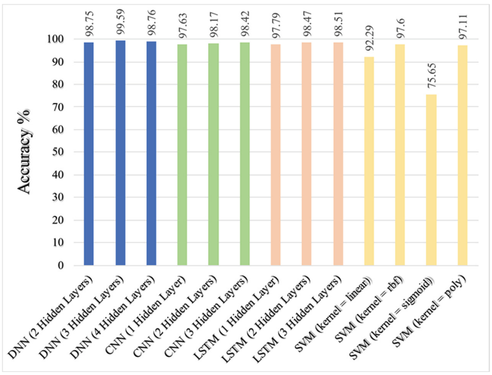

Fig. 6. Performance comparison based on accuracy of DNN, LSTM, CNN and SVM models

Further, the recall, precision, and F1-score values for each of the classifiers taking different hyperparameters each time are shown in Table 1.

Table 1. Performance comparison of DNN, LSTM, CNN and, SVM models with different configurations on the created dataset.

Model	Training Time (s)	Prediction Time(s)	Accuracy %	Precision		Recall		F1-Score	
				Normal	Covert	Normal	Covert	Normal	Covert
DNN (3 Hidden Layers)	54.76	0.30	**99.59**	**0.99**	**1.00**	**1.00**	**0.99**	**1.00**	**1.00**
DNN (2 Hidden Layers)	52.81	0.22	98.75	0.98	**1.00**	**1.00**	0.97	0.99	0.98
DNN (4 Hidden Layers)	59.20	0.32	98.76	0.98	**1.00**	**1.00**	0.97	0.99	0.98
LSTM (1 Layer)	334	0.93	97.79	0.96	**1.00**	**1.00**	0.95	0.98	0.97
LSTM (2 Layers)	634.38	1.69	98.47	0.98	**1.00**	**1.00**	0.97	0.99	0.98
LSTM (3 Layers)	955.28	2.76	98.51	0.98	0.99	**1.00**	0.97	0.99	0.98
CNN (1 Layer)	123.70	0.30	97.63	0.96	**1.00**	**1.00**	0.94	0.98	0.97
CNN (2 Layers)	168.69	0.37	98.167	0.97	**1.00**	**1.00**	0.96	0.98	0.98
CNN (3 Layers)	312.58	0.48	98.42	0.97	**1.00**	**1.00**	0.96	0.99	0.98
SVM (Kernel = linear)	4.34	0.23	92.29	0.89	0.98	0.99	0.84	0.94	0.90
SVM (Kernel = rbf)	**0.617**	0.39	97.6	0.96	**1.00**	**1.00**	0.94	0.98	0.97
SVM (Kernel = sigmoid)	3.17	0.55	75.65	0.75	0.76	0.86	0.59	0.80	0.66
SVM (Kernel = poly)	0.94	**0.158**	97.11	0.96	0.99	**1.00**	0.94	0.98	0.96

The proposed DICCh-D (DNN with 3 hidden layers) took slightly more time than some of its counterparts but in terms of accuracy it clearly outperformed the DNN algorithm with 2 and 4 hidden layers, and all CNN, LSTM and SVM configurations in consideration. It showed an accuracy of 99.59% on the testing dataset, and recorded a precision and a recall value of 1.00 and 0.99 for identifying the IPv6 packets carrying covert data.

Table 2. Comparison of DICCh-D with the existing technique

Model	Training Time (s)	Accuracy(%)	Precision	Recall
Naïve Bayes by Salih et al [19]	**0.15**	94.47%	0.960	0.985
Proposed DICCh-D	52.81	**99.59%**	**1.00**	**0.99**

Salih et al. used a Naive Bayes classifier to detect IPv6-based covert channels with an accuracy of 94.47% [19]. They proposed a framework that uses a hybrid feature selection technique using Intelligent Heuristic Algorithm (IHA) in addition to a modified Decision Tree C4.5 to create primary training data to detect hidden channels in the IPv6 network. A comparison of DICCh-D with the same, in terms of training time, accuracy, precision, and recall is shown in Table 2. Although DICCh-D takes a longer time to train in comparison to the framework proposed by Salih et al., it surely outperforms the same in terms of accuracy, precision, and recall values.

6 Conclusion

In this paper, DICCh-D, a model that detects IPv6-based covert channels using a DNN has been proposed. The dataset needed to train, validate and test DICCh-D was developed using normal IPv6 packets taken from CAIDA's Anonymized Internet Traces 2019 dataset and the covert IPv6 packets. The covert IPv6 packets were generated using the pcapStego tool for hiding data in the Traffic Class and Flow Label fields in the headers of IPv6 packets. Moreover, for testing the generalization ability of the proposed DICCh-D, a testing dataset was kept aside before the model creation and validation phase. The proposed DICCh-D obtained an accuracy of 99.59% and precision and recall rate of 1.00 and 0.99 respectively for identifying IPv6 packets with covert data in the testing dataset. When compared with the state-of-the-art Deep Learning methods like CNN and LSTM, DICCh-D gave better results in terms of time taken for training and testing with comparable accuracy, precision, and recall values. The proposed DICCh-D also outperformed SVM in terms of accuracy.

References

1. Mazurczyk, W., Caviglione, L.: Information hiding as a challenge for malware detection. IEEE Secur. Priv. **2**(13), 89–93 (2015)
2. Lucena, N.B., Lewandowski, G., Chapin, S.J.: Covert channels in IPv6. In: Danezis, G., Martin, D. (eds.) PET 2005. LNCS, vol. 3856, pp. 147–166. Springer, Heidelberg (2006). https://doi.org/10.1007/11767831_10

3. Mazurczyk, W., Powojski, K., Caviglione L.: IPv6 covert channels in the wild. In: Proceedings of the Third Central European Cybersecurity Conference, pp. 1–6 (2019)
4. The CAIDA UCSD Anonymized Internet Traces Dataset - [20 Jan 2019, 21 Jan 2019], Center for Applied Internet Data Analysis (2021). https://www.caida.org/data/passive/passive_dataset. Accessed 20 Dec 2021
5. Zuppelli, M., Caviglione, L.: pcapStego - a tool for generating traffic traces for experimenting with network covert channels. In: The 16th International Conference on Availability, Reliability and Security, pp. 1–8 (2021)
6. Deering, S., Hinden, R.: Internet Protocol, Version 6 (Specifications). Internet Engineering Task Force. https://tools.ietf.org/html/rfc8200#section-6. Accessed 10 Dec 2021
7. Google: Google IPv6 (2022). https://www.google.com/intl/en/ipv6/statistics.html#tab=ipv6-adoption. Accessed 15 Jan 2022
8. Handel, T.G., Sandford, M.T.: Hiding data in the OSI network model. In: Anderson, R. (ed.) IH 1996. LNCS, vol. 1174, pp. 23–38. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61996-8_29
9. Bedi, P., Dua, A.: Network steganography using the overflow field of timestamp option in an IPv4 packet. Procedia Comput. Sci. **171**, 1810–1818 (2020)
10. Giffin, J., Greenstadt, R., Litwack, P., Tibetts, R.: Covert messaging through TCP timestamps. In: International Workshop on Privacy Enhancing Technologies (2002)
11. Sabeti, V., Shoaei, M.: New high secure network steganography method based on packet length. ISC Int. J. Inf. Secur. **12**(1), 24–44 (2020)
12. Bedi, P., Dua, A.: ARPNetSteg: network steganography using address resolution protocol. Int. J. Electron. Telecommun. **66**(4), 671–677 (2020)
13. Dua, A., Jindal, V., Bedi, P.: Covert communication using address resolution protocol broadcast request messages. In: 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), pp. 1–6. IEEE (2021)
14. Ray B., Mishra S.: Secure and reliable covert channel. In: 4th Annual Workshop on Cyber Security and Information Intelligence Research: Developing Strategies to Meet the Cyber Security and Information Intelligence (2008)
15. Bedi, P., Dua, A.: Network steganography using extension headers in IPv6. In: Badica, C., Liatsis, P., Kharb, L., Chahal, D. (eds.) ICICCT 2020. CCIS, vol. 1170, pp. 98–110. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-9671-1_8
16. Lewandowski, G., Lucena, N.B., Chapin, S.J.: Analyzing network-aware active wardens in IPv6. In: Camenisch, J.L., Collberg, C.S., Johnson, N.F., Sallee, P. (eds.) IH 2006. LNCS, vol. 4437, pp. 58–77. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74124-4_5
17. Caviglione, L., Mazurczyk, W., Zuppelli, M., Schaffhauser, A., Repetto, M.: Kernel-level tracing for detecting stegomalware and covert channels in Linux environments. Comput. Netw. **191**, 108010 (2021)
18. Repetto, M., Caviglione, L., Zuppelli, M.: bccstego: a framework for investigating network covert channels. In: The 16th International Conference on Availability, Reliability and Security, pp. 1–7 (2021)
19. Abdulrahman, S., Ma, X. Peytchev, E.: Detection and classification of covert channels in IPv6 using enhanced machine learning. In: Proceedings of International Conference on Computer Technology and Information Systems (2015)
20. Senaid A., Rashid F.: A deep learning-based approach to detect covert channels attacks and anomaly in new generation internet protocol IPv6. Master's thesis (2020)
21. Wireshark. https://www.wireshark.org. Accessed 29 Dec 2021