IJCSI International Journal of Computer Science Issues, Volume 15, Issue 5, September 2018
ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784
www.IJCSI.org https://doi.org/10.5281/zenodo.1467646

Detecting Distributed Denial of Service Attacks Using Hidden Markov Models

Sulaiman Alhaidari[1], Ali Alharbi[2] and Mohamed Zohdy [3]

[1] School of Engineering and Computer Science, Oakland University, Rochester, MI 48309, U.S.A

[2] School of Engineering and Computer Science, Oakland University, Rochester, MI 48309, U.S.A

[3] School of Engineering and Computer Science, Oakland University, Rochester, MI 48309, U.S.A

Abstract

Distributed Denial of Service (DDoS) attacks considered the most critical attack for cyber security and serious security threat to Internet services in recent years. These attacks have evolved to be *increasingly* sophisticated, complex, and difficult to mitigate and detect. In this paper, we propose a new approach using HMM to detect DDoS attacks. The performance of the proposed approach is generally better and achieve higher detection rate and lower false positive rate comparing with two other machine-learning algorithms Naive Bayes and Neural Network. Training and testing applied on a DDoS data set with reduced feature. Using the reduced feature set after applying the Feature Pruning algorithm that we implemented obtains a significant improvement in detection performance and reduction model training and testing time.

Keywords: - *Hidden Markov models (HMM), distributed denial of service (DDoS).*

1. Introduction

Computer networks are vital to the smooth operation of the global society and economy. The three parallel control objectives of confidentiality (or secrecy), integrity (or correctness), and availability must be assured to the maximum extent possible. Otherwise, our entire electronic infrastructure—travel, power, logistics, even defense—can almost literally disintegrate. Malware that takes advantage of distributed denial of service techniques is capable of compromising these three control objectives. It is therefore important to understand the various DDoS attacks that can be perpetrated against computer systems and networks. In this paper, we investigate the DDoS attacks and then examine the applicability of two competing analytical methods—the hidden Markov model (HMM) and machine learning (ML)—in protecting systems and networks against them.

DDoS attacks are intended to make it impossible for authorized users of computer systems to access the resources of those systems. In general, DoS attacks and their distributed analogues operate by flooding target hosts and networks with traffic that is orchestrated to waste resources on the target hosts, thereby inducing them either to crash or to enter an unresponsive state [1]. The two most critically important DDoS attacks are the Smurf attack and the SYN flood attack. In the Smurf attack, the attacker synthesizes phony traffic that consists of ICMP broadcast packets. The source IP address recorded in these packets is spoofed so that they appear to originate from a host located within the trusted intranet under attack. By actually responding to the packets, the target host becomes successively more burdened with fabricating more and more responses until it is altogether unable to muster the resources to process any more network traffic. The Smurf attack can be readily defended against, albeit, by implementing a firewall that is able to recognize malicious inbound packets as Martians, that is, packets that arrive at the wrong place, seemingly as the result of a routing error [3]. Assuming that the firewall is built atop a dual-homed host with two network interfaces, it is straightforward to recognize and discard arriving packets that claim to be from hosts within the network but are being received at the interface physically connected to the outbound side [15]. The SYN flood attack is conceptually similar. In this class of attack, requests to establish brand-new TCP connections arrive at a rapid rate. These requests take the form of initiations of the so-called three-way handshake. By starting such handshakes but never completing them—

merely sending more and more handshake initiations—the attacker succeeds in exhausting the pool of system resources available to the server for receiving and processing further inbound connections. Therefore, all network processing grinds to a halt ("Understanding Denial of Service Attacks").

The reminder of the paper is organized as follows: Section 2 reviews the related works. Section 3 presents the background of HMM and its problems. Section 4 discusses our methodology and approach. Section 5 discusses evaluation metrics and our results. Finally, Section 6 concludes our paper.

2. Related Works

Research efforts to date that target the detection of and defense against DDoS attacks have taken advantage of both the hidden Markov model (HMM). Four studies that have applied the hidden Markov model to DDoS attacks are those by Jain & Abouzakhar; Bhole & Patil; Devarakonda et al.; and Khosronejad et al.

Jain & Abouzakhar analyzed the performance of the HMM in intrusion detection systems (IDS), including the perpetration of DDoS attacks, through the design of what they termed a Support Vector Machine (SVM). The machine was capable of distinguishing and classifying TCP services that include both "normal" and "abnormal" packets, including the improperly formed packets that participate in the Smurf attack. They used the KDD CUP 1999 dataset to underlie their research and serve as the training set for the SVM. The authors demonstrated that the hidden Markov model was able properly to classify network traffic with accuracy ranging from 76 to 99 percent, depending upon the precise circumstances [8].

Bhole & Patil used HMM to construct an intrusion detection engine that combined the detection of anomalies with the more traditional approach of signature detection. The HMM enabled them to construct an engine that was particularly effective in the recognition of novel, that is, previously unseen attacks. Although they found that signature-based detection was more efficient and timely for known attacks, HMM enabled new classes of attacks to be learned [2].

Devarakonda et al. decided to augment a traditional intrusion detection system by using a combination of a Bayesian network and a hidden Markov model. The IDS framework was designed to incorporate various levels of processing, including learning from the training data by subjecting it to the HMM and allied Bayesian classifier. The work was completed using the KDD CUP dataset and demonstrated "performance of high order" [4].

Finally, Khosronejad et al. compared two standard approaches to IDS, the well-known C5.0 model and the HMM, also combining the two into what they termed "a hierarchical hybrid intelligent system model." Empirical results established that the hybrid system delivered considerable accuracy when applied to the KDD CUP 99 dataset [10].

3. Hidden Markov Model

HMMs have been extensively utilized in many applications such as speech recognition, finance, computer vision and bioinformatics. HMM is composed of hidden states, and observable emissions. States are the desirable events in a system, which are not visible to the observer, while emissions are the observable symbols emitted from the states. Using a sequence of emissions, an HMM can predict whether a system is in each state at a certain time. Fig.1 shows the first-order HMM, where the observations are shaded in gray.

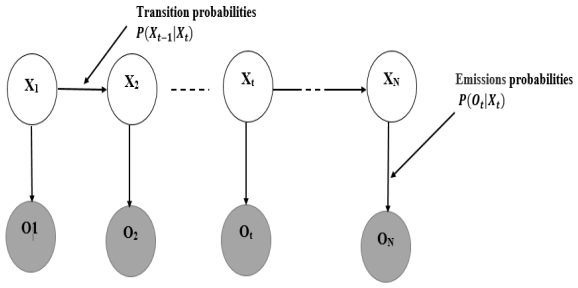

Fig. 1 First-order HMM, where the observations are shaded in gray

There are three fundamental sub-problems to HMM. The first is the evaluation problem. This addresses calculating the probability that the model can generate the indicated output sequence. The second is the decoding problem. This strives to derive the model history—that is, sequence of states—that was most likely responsible for the generation of a specified output sequence. The third is the so-called learning problem. This problem endeavors to deduce model parameters from a set of output sequences in a manner that offers the greatest fidelity, that is, likelihood of correct sequence generation.

3.1. The evaluation problem

The evaluation problem is the first of these to be analyzed. Both a forward algorithm and a backward algorithm are integral to this problem. The forward algorithm determines the various conditional probabilities, or alphas, by forward reasoning. The procedure is subdivided into three phases known as initialization, induction, and termination. The algorithm operates so that it requires only TN^2 operations to evaluate all of the conditional probabilities engendered.

By way of contrast, the backward algorithm derives a set of backward variables, or betas. These values reflect the probabilities of observation of a set of subsets of partial state sequences, or histories. The algorithm requires the completion of two steps, initialization and induction, and—like the forward algorithm—is able to deliver its results in only TN^2 operations [6].

In stricter mathematical detail, the objective of both the forward and backward algorithms is to calculate the probability of an observation history, $O = \{O_1, O_2, \dots, O_T\}$ (1)

for a given model $\lambda = (A, B, \pi)$ (2)

The algorithms carefully examine all possible sequences of states so as to determine the attendant probabilities. According to the forward procedure, the probabilities of occurrence of the partial histories are represented by the set, $\alpha_t(i)$ where,

$$a_{t(i)} = P(O_1, O_{2,}.., O_T, S_N = S_i|\Lambda) \quad (3)$$

Computing this set requires the completion of an initialization and induction phase, in which,

$$\alpha_j(t) = \begin{cases} \pi_j b_{jv(1)}, & t = 1 \\ \left(\sum_{i=1}^{N} \alpha_j(t-1)a_{ij}\right) b_{jo(t)}, & t = 2, \dots, T \end{cases} \quad (4)$$

and a termination phase, in which,

$$P(O|\Lambda) = \sum_{i=1}^{N} \alpha_T(i) \quad (5)$$

The backward procedure instead derives the set { $\beta_t(i)$ }, defined as

$$a_{t(i)} = P(O_{t+1}, O_{t+2,}.., O_T, S_N = S_i|\Lambda) \quad (6)$$

By way of contrast, the backward algorithm requires only the initialization and induction phase phase, in which

$$\beta_j(t) = \begin{cases} 1, & t = T \\ \sum_{i=1}^{N} \beta_i(t+1)a_{ji}\, b_{io(t+1)}, & t = T-1, \dots, 1 \end{cases} \quad (7)$$

Forward Formula

$$P(O^T|\,\lambda) = \sum_{j=1}^{N} \alpha_j(T) \quad (8)$$

Backward Formula

$$P(O^T|\,\lambda) = \sum_{j=1}^{N} \beta_j(1)\pi_j b_{jo(1)} \quad (9)$$

3.2. The decoding problem

The decoding problem is a sub-problem that considers series of observations and tries to determine an optimal path through the hidden state sequence. A dynamic programming technique known as the Viterbi algorithm is typically relied upon in order to calculate the critical path. This algorithm requires $O(NQ^2)$ time completely to establish the path through the network that maximizes the conjoint probability of transit [5].

$$P(S_i|O^T, \lambda) = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)} \quad (10)$$

3.3. The learning problem

The third problem is the learning problem: given a sequence of emissions, how can an HMM be trained which best matches the sequence. This problem is all about maximizing the parameters of the HMM – A, B, and π, – to get the most descriptive model for the system. The algorithm of choice for solving this problem is the Baum-welch algorithm.

This algorithm can be broken into two steps: estimation and update.

Estimation The estimation step of the algorithm first calculates the forward probabilities, then the backward probabilities to find the likelyhood of the observed sequence being produced from the estimated HMM (because of this step, this algorithm is sometimes referred to as the forward-backward algorithm).

Update During this step, the algorithm uses Bayes' theorem to create temporary varables and update the parameters:

$$\gamma_i(t) = P(S_i|\, O^T, \lambda) = \frac{P(w_i, o^T|\lambda)}{P(o^T|\lambda)} = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)} \quad (11)$$

$$\varepsilon_{ij}(t) = P\big(S_t = s_{i,}\, S_{t+1} = S_j \big|\, S^T, \lambda\big) = \frac{P\big(s_i, s_j, O^T \big|\, \lambda\big)}{P(O^T|\lambda)} \quad (12)$$

$$\varepsilon_{ij}(t) = \frac{\alpha_i(t)a_{ij}\beta_i(t+1)b_j(o_{t+1})}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i(t)a_{ij}\beta_j(t+1)b_j(o_{t+1})} \quad (13)$$

As one may notice, $\gamma_i(t)$ is actually the Viterbi algorithm used to get the estimated state sequence. On the other hand, $\varepsilon_{ij}(t)$ is the probability the model is in states i and j at times t and t+1 given the observed sequence V^T. With these, the parameters of the HMM can now be updated:

$$\hat{\pi} = \gamma_i(1) \quad (14)$$

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \varepsilon_{ij}(t)}{\sum_{t=1}^{T} \gamma_i(t)} \quad (15)$$

$$\hat{b}_i(o_k) = \frac{\sum_{t=1}^{T} 1_{o_t=o_k} \gamma_i(t)}{\sum_{t=1}^{T} \gamma_i(t)},$$

$$\text{where } 1_{o_t=o_k} = \begin{cases} 1, & if\ o_t = o_k \\ 0, & otherwise \end{cases} \quad (16)$$

These updated parameters are then fed back into the estimation step and the process repeats iteratively until a desired level of convergence is achieved. Note: It is possible for the algorithm to converge prematurely due to a local maximum, in this overfitting will occur. To counter this, one can adjust the tolerance and train using different initial parameters.

4. Research Methodology

4.1. Dataset

One of the difficulties and Challenges associated with using machine learning is finding a large realistic training dataset. In this study, DDoS dataset is used for the evaluation [19]. It contains 27 features, which are labeled as either normal or an attack as it shown in Table 1. The DDoS dataset include four types of the DDoS attack, which are Smurf, UDP-Flood, HTTP-Flood and SIDDOS. From this data set, a small portion of training and testing data is selected for the experimentation of the model.

Table-1 List of features of DDOS dataset.

Feature #	***Description***	***Feature #***	***Description***
1	SRC ADD	15	PKT IN
2	DES ADD	16	PKTOUT
3	PKT ID	17	PKTR
4	FROM NODE	18	PKT DELAY NODE
5	TO NODE	19	**PKTRATE**
6	PKT TYPE	20	**BYTE RATE**
7	**PKT SIZE**	21	PKT AVG SIZE
8	FLAGS	22	UTILIZATION
9	FID	23	PKT DELAY
10	**SEQ NUMBER**	24	PKT SEND TIME
11	NUMBER OF PKT	25	PKT RESEVED TIME
12	NUMBER OF BYTE	26	FIRST PKT SENT
13	NODE NAME FROM	27	**LAST PKT RESEVED**
14	NODE NAME TO	28	ATTACK /NORMAL

4.2. Feature Pruning

Feature Pruning is a method of eliminating features from the original dataset to obtain a subset of features that has higher accuracy on low-cardinality sets. It plays a key role in building detection models. However, The DDoS dataset has 27 Features, from which reducing features from the full data set will reduce both the data and the computational complexity and improve both the efficiency and the accuracy of the model. On the other hand, using all 27 features without applying Feature Pruning might increase the overhead of the model, which leads to increases the time to build the model. In order to perform feature pruning, we first need to standardize the data and then combine the standardized data to one sequence of observation, which then could be used afterward with Viterbi algorithm to compute the most likely state sequence and then be compared to the actual sequence of states to determine accuracy of the feature. The feature-pruning algorithm that we implemented automates this process and eliminates each feature from the full set of the features, and then checks the accuracy of the subset of features. More features that are least significant are eliminated if the obtained accuracy is within a certain tolerance of the accuracy, equal, or higher than the previous accuracy of every feature combined. This process continues until no improvement of the accuracy is observed on elimination of features. The pseudo-code is presented in Algorithm 1 that shows outlines the steps of method. The features in bold in Table 1 are the significant features that obtained by the Feature Pruning algorithm that used in our detection approach.

Algorithm 1 Feature Pruning Algorithm for HMM

1: Begin
2: feature_pruning (X, L, Already_Cheched)
3: **Input**: X-The number of desired features
L-List of N feature Vectors
Already_Cheched- A set containing already checked combinations of features
4: **Output**: R – The reduced feature set of length X
5: **if** Already_Cheched contains (L) **then**
6: Return

7:**end**

8: **if** N-X=0 **then** // L contains the desired of features so test accuracy

9: Return L, evaluate accuracy ()

10: **else**

11: Ai = {All features in L except feature i} for all i=1,..., N //Create subsets of L with one less feature
12: Return max (feature_ pruning (X,A1), (feature_ pruning (X,A2),.....,(feature_ pruning (X,AN) //Return the subset with best accuracy

13: **end**

14: End

4.3. Initializing HMM Parameters

The next step after obtaining the significant features set and before training an HMM is to initialize the parameters. Technically, the HMM parameters could be initialized random and then determined or estimated over several training iterations by using the Baum-Welch training algorithm (also known as Forward-Backward algorithm). It is necessary to start with a rough guess to determine the parameters of HMM (the transition probability matrix and emission probability matrix). Once they are determined, they can be re-estimated by applying the Baum-Welch algorithm and find the more accurate parameters and obtain the HMM best describes the observed sequence. Then, this trained HMM can be used to the testing set to ensure it is able to detect the proper states. Fig. 2 shown the flow chart of the proposed approach.

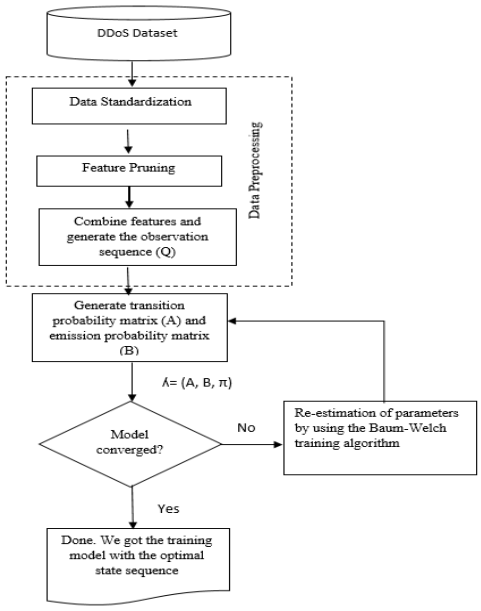

Fig. 2 Flow chart of training algorithm for HMM

5. Performance analysis and evaluation

5.1. Method of performance testing

To evaluate the performance of our proposed model and how accurate the model classifying and predicting the class label of attack and normal, we need to know the following four terms: True Positive (TP): The number of attacks instances classified as attacks. True Negative (TN): The number of non-attacks instances classified as non-attacks . False Negative (FN): The number of attacks instances classified as non-attacks. False Positive (FP): The number of non-attacks instances classified as attacks

Confusion matrix for a two class case (Attack and non-attacks) shown in Table 2

Table-2 Confusion matrix for a two-class case (Attack and non-attacks)

		Predicted Class	
		Attack	Non-attacks
Actual Class	Attack	TP	FN
	Non-attacks	FP	TN

For this study, we used the following performance measures to test the performance of the proposed model:

Accuracy: the ratio of the total number of correctly predicted instances to total number of all instances. In our study, accuracy measures by using the Viterbi algorithm to generate a likely state sequence and compare it to the known state sequence to get TP, FP, FN, and TN. The accuracy can be calculated by using the following equation:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (17)$$

The error rate (misclassification rate)/ False Negative Rate (FNR): the ratio of the total number of misclassifications to total number of all predictions. The error rate can be calculated by using the following equation:

$$Error\ rate = \frac{FP + FN}{TP + FP + TN + FN} \quad (18)$$

Fall-out/ False Positive Rate (FPR): the ratio of the number of detected false positives to total number of predictions. The Fall-out can be calculated by using the following equation:

$$Error\ rate = \frac{FP}{FP + TN} \quad (19)$$

The sensitivity/ True Positive Rate (TPR): the ratio of the total number of detected true positive that are correctly identified as attack to total number of positive instances. The Sensitivity can be calculated by using the following equation:

$$\text{sensitivity} = \frac{TP}{P} \quad (20)$$

Where,
P is the number of positive instances, P=TP+FP.

The specificity/ True Negative Rate (TNR): the ratio of the total number of detected true negative that are correctly identified as non-attacks to all the negative instances. The Specificity can be calculated by using the following equation:

$$specificity = \frac{TN}{N} \quad (21)$$

Where,
N is the number of negative instances, N=TN+FN.

The precision /Positive Predictive Value (PPV) and recall: The precision and recall measures are widely used for performance evaluation of machine-learning

classification methods. Precision is the ratio of the total number of positive instances that are correctly identified as attack to the total number of attacks. Whereas recall is the ratio of the total number of instances that are correctly identified as attack to the total number of all the instances that correctly identified as attack and misidentified attacks (it is the same as sensitivity). The precision and recall can be calculated by using the following equations

$$Precision = \frac{TP}{TP + FP} \quad (22)$$

$$Recall = \frac{TP}{TP + FN} \quad (23)$$

F measure: F measure is a testing score that testing the accuracy of the model and it considers both the precision and recall. F measure can be calculated by using the following equation

$$F\ meaure = 2 * \frac{Precision * recall}{Precision + reacall} \quad (24)$$

Roc curve: One of the prime benefits of using an ROC curve is to get the ability to notice the tradeoff between the true positive rate (sensitivity) and false positive rate (1-specificity) for all possible cut off points (thresholds) rather than just one cut off point. The area under the ROC curve (AUC) measures the ability of the model correctly distinguish between classes (Attack or Non-attack). Fig 3 shown the ROC Curves for the performance of algorithms HMM, Naive Bayes and Neural Network.

5.2. Result Evaluation

The result shows our proposed approach can obtain better results in terms of attack detection rate. Moreover, the result shows improved performance with a reduced feature set after applying the Feature Pruning algorithm and selected the most important features. By training an HMM and testing it on the DDoS set, attacks were detected with greater than 97 percent accuracy in most trial runs. Table 4 summarizes the results of an experiment. The performance of the HMM algorithm compares against two classification algorithms, Neural Network and Naive Bayes algorithm. Both were taken from the WEKA. Fig. 4 shows in a graphical way a comparison between the three classification algorithms.

Table-4 the summarily of the experiment results

Performance Measures /Classification Algorithm	***Training/Testing (70/30 %)***		
	HMM	***Naive Bayes***	***NN***
Accuracy	0.9741	0.9348	0.9356
Error rate	0.0259	0.0652	0.0644
Fall-out	0.0104	0.1888	0.1623
Sensitivity/ recall	0.8413	0.9431	0.9418
specificity	0.9896	0.8112	0.8377
precision	0.9038	0.9867	0.9893
F measure	0.8714	0.9644	0.9649
Area Under ROC (AUC)	0.9334	0.9177	0.8624

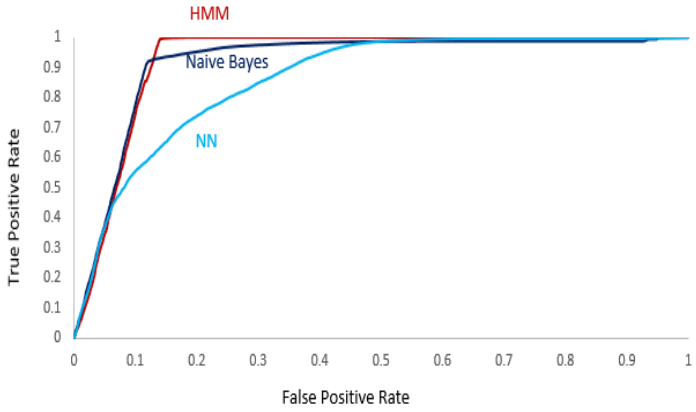

Fig 3 ROC curves for the performance of classification algorithms HMM, Naive Bayes and Neural Network.

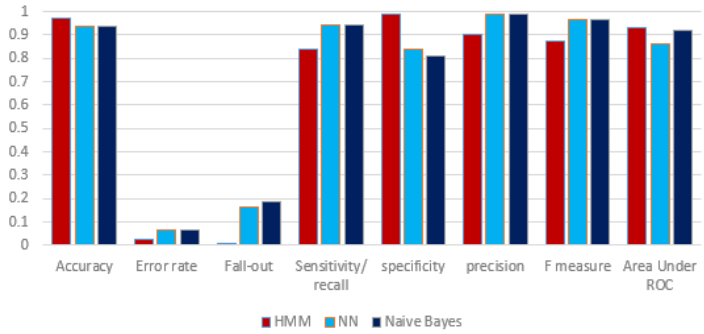

Fig. 4 Comparison chart of the performance of algorithms HMM, Naive Bayes and Neural Network.

6. Conclusion

In recent years, machine-learning methods are gaining the most attention in prediction due to its ability to learn, evolve, improve and adapt. Thus, in this paper, we presented our detection approach using Hidden Markov Models (HMM) that applied and tested on the DDoS dataset to detect the DDoS attack. The result shows that we able to produce a great performance with maximum accuracy, minimum error rate and False Positive Rate. The detection result demonstrated that HMM gives a more accurate result than would have been obtained by Neural Network and Naive Bayes algorithms while detecting the attacks. It achieved 97.41 % accuracy.

References

[1] Millan, M., DeJonge, A., Alharbi, A., Alshammari, A., Alhaidari, S., Alshammari, A., Zohdy, M. (2018). Application of Extended Hidden Markov Models to detect threats in IoT. International Journal of Computer and Information Technology, 7(5), 2279 – 0764.

[2] Bhole, Ashish T., & Patil, Archana I. "Intrusion Detection with Hidden Markov Model and WEKA Tool." International Journal of Computer Applications 85.13 (January 2014). Retrieved from https://pdfs.semanticscholar.org/2bc9/3fed20ffff4ad6da55e0345a66b986d5de9e.pdf.

[3] Comer, Douglas E. Internetworking with TCP/IP: Principles, Protocols, and Architecture. Prentice Hall, 2000.Conrad, Eric, et al. CISSP Study Guide. Elsevier, 2012.

[4] Devarakonda, Nagaraju, et al. "Intrusion Detection System Using Bayesian Network and Hidden Markov Model." Procedia Technology 4 (2012). Retrieved from https://www.sciencedirect.com/science/article/pii/S2212017731200360X.

[5] Freitas, Ana Teresa. "Hidden Markov Models." University of Lisbon, 2011. Retrieved from https://fenix.tecnico.ulisboa.pt/downloadFile/377957732693 2/Modelos_prob_12.pdf.

[6] Gutierrez-Osuna, Ricardo. "L23: Hidden Markov Models." Texas A&M University. Retrieved from http://research.cs.tamu.edu/prism/lectures/pr/pr_l23.pdf.

[7] Haq, Nutan Farah, et al. "Application of Machine Learning Approaches in Intrusion Detection System: A Survey." International Journal of Advanced Research in Artificial Intelligence 4.3 (2015). Retrieved from https://thesai.org/Downloads/IJARAI/Volume4No3/Paper_2-Application_of_Machine_Learning_Approaches_in_Intrusion_Detection_System.pdf.

[8] Jain, Rachi, & Abouzakhar, Nasser S. "A Comparative Study of Hidden Markov Model and Support Vector Machine in Anomaly Intrusion Detection." Journal of Internet Technology and Secured Transactions 2.3 (September 2013). Retrieved from https://pdfs.semanticscholar.org/6aef/e685dc5cd82af9e083c68b76e346042946b8.pdf.

[9] Jurafski, Daniel, & Martin, James H. Speech and Language Processing. Prentice Hall, 2016. Retrieved from https://web.stanford.edu/~jurafsky/slp3/9.pdf.

[10] Khosronejad, Mahsa, et al. "Developing a Hybrid Method of Hidden Markov Models and C5.0 as an Intrusion Detection System." International Journal of Database Theory and Application 6.5 (2013). Retrieved from http://www.sersc.org/journals/IJDTA/vol6_no5/15.pdf.

[11] Paliwal, Swati, & Gupta, Ravindra. "Denial-of-Service, Probing & Remote-to-User Attack Detection Using Genetic Algorithm." International Journal of Computer Applications 60.19 (December 2012). Retrieved from https://pdfs.semanticscholar.org/060d/0c18c3f490720b62e40e7003aa7f75d50941.pdf.

[12] Revathi, S., & Malathi, A. "Detecting User-to-Root (U2R) Attacks Based on Various Machine Learning Techniques." International Journal of Advanced Research in Computer and Communication Engineering 3.4 (April 2014). Retrieved from https://ijarcce.com/wp-content/uploads/2012/03/IJARCCE4J-a-revathi-EPCglobal-Gen-2-RFID.pdf.

[13] Shmatikov, Vitaly, & Wang, Ming-Hsiu. "Security Against Probe-Response Attacks in Collaborative Intrusion Detection." Cornell University. Retrieved from https://www.cs.cornell.edu/~shmat/shmat_lsad07.pdf.

[14] Tsai, Chih-Fong, et al. "Intrusion Detection by Machine Learning: A Review." Expert Systems with Applications 36.10 (December 2009). Retrieved from https://www.sciencedirect.com/science/article/pii/S0957417409004801."Understanding Denial of Service Attacks." United States Computer Emergency Readiness Team, 28 June 2018. Retrieved from https://www.us-cert.gov/ncas/tips/ST04-015.

[15] Wing. "7 Different Types of Firewalls." Security Wing, 12 September 2012. Retrieved from https://securitywing.com/types-of-firewall/.

[16] Wood, Patrick H., & Kochan, Stephen G. UNIX System Security. Hayden Book Company, 1985.

[17] Yemini, Yechiam. "Chapter 4: Hidden Markov Models." Columbia University. Retrieved from http://www.cs.columbia.edu/4761/notes07/chapter4.3-HMM.pdf

[18] Duda, R., Hart, P., & Stork, D. Pattern Classification. 2nd ed., John Wiley and Sons Inc., 2001.

[19] Alkasassbeh, Mouhammd, et al. "Detecting distributed denial of service attacks using data mining techniques." International Journal of Advanced Computer Science and Applications 7.1 (2016).