Are Machine Learning Based Intrusion Detection System Always Secure?
An Insight into Tampered Learning

Rupam Kumar Sharma\textsuperscript{a}, Hemanta Kr Kalita\textsuperscript{b} and Biju Issac\textsuperscript{c}

\textsuperscript{a,b}Department of Information Technology, NEHU, Shillong, Meghalaya, India
\textsuperscript{c}School of Computing, Media & the Arts, Teesside University, England, UK

Abstract. Machine learning is successful in many applications including securing a network from unseen attack. The application of learning algorithm for detecting anomaly in a network has been fundamental since few years. With increasing use of machine learning techniques, it has become important to study to what extent it is good to be dependent on them. Altogether a different discipline called ‘adversarial learning’ have come up as a separate dimension of study. The work in this paper is to test the robustness of online machine learning based IDS to carefully crafted packets by the attacker called poison packets. The objective is to observe how a remote attacker can deviate the normal behavior of machine learning based classifier in the IDS by injecting the network with carefully crafted packets externally, that may seem normal by the classification algorithm and the instance made part of its future training set. This behavior eventually can lead to a poisoned learning by the classification algorithm in the long run, resulting in misclassification of true attack instances. This work explores one such approach with SOM and SVM as the online learning-based classification algorithms.

Keywords: Adversarial learning; Machine learning; Poison learning; Intrusion Detection System; Artificial Intelligence, NSL-KDD Dataset, SVM, support vectors.

1. INTRODUCTION

Intrusion Detection and Prevention systems (IDS/IPS) are one of the critical components of the network of an organization or an institution. Even though IDS involving machine learning have not been of much practical considerations in a real network but still they have proven effective to withstand future unseen attacks. Much of the research work have also been focused on detecting online network attacks apart from detecting off line attacks by analyzing the log data or offline data. Till date several IDS systems are designed and developed based on many different machine learning techniques. Most of these techniques are used as a classifier to normal and attack packets. Literature study also portrays that some IDS are based on single learning techniques such as Genetic Algorithm, Artificial Neural Network etc, while most others involve multiple learning involving the process of ensemble techniques. However, the accuracy of such learning algorithms depends on the type and amount of training data considered. Bio inspired algorithms are also coming up in recent times [48,49,53]. Recently online statistical machine learning has also become an important and useful approach to IDS. In such cases the learning is periodically retrained on the online data for better classification results i.e. every new incoming packet is initially classified by the classifier either as normal or anomaly. If the packet turns out to be normal than it becomes part of future training set. This behavior of learning has been exploited by adversaries very well. The adversaries with minimum knowledge of the training data set used crafted data in such a way that the classifier may treat it as normal but in the long run may lead to a poison attack. In this paper the proposed model of online IDS by Lee, Seungmin, Gisung Kim et.al [1] have been adopted as a part of study due to high accuracy claim and is tested on NSL KDD data set [2]. The model was later subjected to poison learning and results were analyzed.

The outline of this paper is as follows. Section 2 outlines different machine learning techniques used in IDS. Section 3 outlines challenges of using machine learning. Section 4 outlines the taxonomy of attacks against IDS. Section 5 outlines the referred model. Section 6 outlines the proposed framework and algorithm. Section 7 discusses the experimental setup, results and analysis. Section 8 proposes a mathematical equation representation corresponding to the number of crafted poison instances. Section 9 discuss the class imbalance consideration followed by Section 10 that discuss the proposed solution that addresses the presented problem and finally followed by conclusion in Section 11.

2. Popular machine learning techniques used in IDS

2.1 Artificial Neural Network

Artificial Neural Network is information processing unit which mimic the neurons of human brain [3]. An Artificial Neural Network consists layer of neurons categorized into input, hidden and output layer [4]. The neural network IDS trained on KDD data set have following three phases [5].

a) Automated parsers to transform raw TCP/IP data into set of vector values fed as input to the neural model.

b) Training: Neural Network model is trained on different network ‘normal’ and ‘attack’ values. Input corresponding to KDD data set have 41 features and the output corresponds to either attack (22 different types) or normal.
c) Testing: Validation on the Test Data for further enhancing the neural model for better classification. Different validation technique such as k-cross validation is adopted at different times.

Some of the recent work using Artificial Neural Network can be found in the following papers [14,15,16].

2.2 Support Vector Machines

Developed by Cortes & Vapnik originally for learning two class discriminant functions from a set of training examples. SVM basically features the following [6,7].

a) Class separation: Seek for the optimal plane that separates the points of the two planes also known as support vectors by maximum distance.

b) Overlapping classes: The influence of data points falling on the wrong side of the planes are weighted down.

c) Non-linearity: The data points that cannot be distinctly separated linearly are transformed into a higher dimensional plane where they become separable.

d) Problem Solution: Representing the entire task as quadratic optimization problem that becomes solvable by some known techniques.

Some of the recent work using SVM in IDS can be found in the following papers [17,18,19].

2.3 Self Organizing Map

This learning is inspired from biological neural model like that of ANN. However, it involves both competitive and correlative learning [8]. Whenever an input is presented to the network model, the neurons compete among themselves and the neuron with closest similarity claims the input and becomes the winner. The winner strengthens his weight with the input. This mechanism spreads to neighbors in Gaussian distribution. The core objective is to reduce the dimension of data visualization. Some of the recent work using SOM In IDS can be found in the following papers [20,21,22].

2.4 Decision Trees

Given a set of instances, Decision tree classify the instances by sorting them down the tree starting from the root and ending in a leaf of the tree. An attribute of an instance is represented as a node of the tree and each branch descending from the node corresponds to one of the possible values of the attribute. This type of learning is mostly used in cases where instances can be represented by set of attribute and value pairs, the output of the target function is not continuous and map to a discrete set of values, considerations of possible errors in the training set and missing values in the training set[9].Some of the recent work using Decision Tree in IDS can be found in the following papers [23,24,25].

2.5 Naïve Bayes Classifier

Naïve Bayes Classifier is a probabilistic classifier. This type of classifier outputs a value p(y|x), i.e. probability of y given x. The computation can be done in two ways.

Firstly, learning and applying the function that computes the class posterior (y|x) and this is called a discriminative process, because given set of instances it discriminates between different classes. The other alternative is to learn the class conditional density p(x|y) for each value of y and to learn the class priors p(y), then one can apply the Bayes rule to compute the posterior [10]. The above is called generative model because for each possible class y, the feature vector x is generated. The advantage of using classifiers with probabilistic output are “reject option”, where the classification is refused if the prediction is uncertain, “changing utility function”, where risk can be minimized by combining the probability distribution with an utility function, “compensating for class imbalance”, where one class is rare than the other(scaled likelihood trick).Some of the recent work using Naïve Bayes in IDS can be found in the following papers [26,27,28].

\[
p(y|x) = \frac{p(x|y)p(y)}{p(x)} = \frac{\sum_{y'} p(x|y')p(y')}{p(x)}
\]

2.6 Fuzzy Logic

Fuzzy logic uses a membership function to indicate degree of belonging of an attribute to a more than one class. It is difficulty to draw a strict boundary between normal and attack and hence instances can be assigned varying degree of normal or attack and for this reason fuzzy is a big choice for designing Intrusion Detection System. With fuzzy it becomes possible to model small deviations to keep false positives/negatives small. The generic form of the fuzzy rule can be represented as follows

IF condition THEN conclusion [weight].

Condition is fuzzy expression defined using fuzzy logic operators fuzzy AND etc, conclusion is an atomic expression and weight is a set of real number [0,1], that portrays the confidence of the rule [11]. Some of the recent work using Fuzzy systems in IDS can be found in the following papers [29,30,31].

2.7 Radial Basis Function

Radial Function are altogether a different type of function where the response decreases or increases monotonically with distance from a point of reference or central point.
One example of such function is Gaussian as shown below.

\[ h(x) = \exp\left(-\frac{(x-c)^2}{2r^2}\right), \text{ where } c \text{ is the center and } r \text{ is the} \]

radius.

Radial basis function network (RBF) are associated with radial functions as shown below in the figure 1 [12]. Some of the recent work using Radial Basis Function in IDS can be found in the following papers [32,22,34].

2.8 K Means Clustering

This algorithm is used to classify objects into ‘k’ number of clusters, based on common features of the objects. The similarity value is computed by considering and minimizing the sum of squares of distances between data points and the corresponding cluster centroid [13]. Some of the recent work using k Means clustering in IDS can be found in the following papers [35,36,37].

3. CHALLENGES IN USING MACHINE LEARNING

Machine Learning has proved to be result promising and many companies such as Amazon uses machine learning for meeting different objectives. However, the success of using machine learning depends on lots of factors of which few are listed below.

3.1 Training Data (Explicit and Implicit)

Training data used in a learning algorithm can be broadly newly categorized into implicit feedback data and explicit feedback data. In explicit feedback data, feature vector corresponding to a message packet is explicitly confirmed as an attack or normal without much difficulty, and correspondingly used to train the learning algorithm. However, in implicit feedback, data features might not be possible to immediately be classified as normal or anomaly because more attributes value might resemble a normal data but overall feature vector or set of features vector might correspond to an anomaly. Such “critical tag” need to be considered with utmost care.

3.2 High Cost Errors

Running an IDS with even a very small rate of false classification might come with high risk to the organization or institution. Falsely classified as Negative might end up in a remote machine gaining access to the internal network and thereby rendering the entire network nonfunctional. The objective would be to design learning algorithms that could ideally make “False Positive” and “False Negative” parameters approximately approach to zero value.

3.3 Rule Generation

For a message or for a given source whose feature vector is classified as abnormal it is critical to judge whether the abnormality corresponds to an attack or a behavior deviating from normal but not an attack. More critical in such cases is automatic rule generation corresponding the feature set of the message or originating source.

3.4 Proper interpretation of traffic over time.

The variability in the network traffic parameters such as volume of traffic, bandwidth consumption, duration of connections, number of connections can make things more critical in operational environment. Adding to the mentioned facts diversity can also be on the application parameters of the messages, nature of protocols and attribute values of different headers fields. Question arises here is the duration for which a given connection or the network should be monitored or how long duration traffic should be aggregated for evaluation. Application layer DoS attack occurs in slow rate and don’t generate massive amount of traffic.

3.5 Data set Hindrance.

The data set that are publicly available such as KDD Cup 1999, NSL-KDD [38,39] are almost a decade old. Learning algorithms are still trained on these existing old data sets which fails to incorporate feature vector of recent attacks such as RUDY[R-U-Dead-Yet]. The alternative could be repository of self-monitored network. However, this could be a complicated task due to non-accessibility to an appropriately sized network.

4. ATTACKS AGAINST MACHINE LEARNING BASED IDS

Even though Machine Learning algorithms have been successful in proving better results, however they are never always secure [59]. An adversary might always seek to explore loopholes for rendering the learning by the algorithm futile. The following outlines properties for analyzing attacks against Machine Learning based IDS as discussed in [41,54].

A. Influence

(a) Causative
(b) Exploratory

B. Security Violation

(a) Integrity
(b) Availability
(c) Privacy

C. Specificity

(a) Targeted
(b) Indiscriminate

The entire model of securing learning algorithms can be framed as a game between the attacker and the learning model. The attacker can poison the learning by manipulating the training instances.

Causative Attack: In this type of attack the adversary influences the training instances [60]. The degree of influence over the attributes of the data may vary based on the amount of access an attacker might have. If the attacker is aware of the truth that online instances are considered by the learning for evolution, he can exploit this fact and frame instances accordingly to gradually deviate the learning towards miss classification. ‘Allergy’ attack, ‘Red herring’ attacks are few to be mentioned.

Exploratory Attack: In this type of attack, the attacker crafts intrusions to successfully evade the classifier. Here the direct influence on the classifier is not performed. Here the attributes
of normal traffic are exploited to form attack vector mimicking a normal vector. If the newly framed vector is successful in evading the classifier, then therein lies the consequences. It might so happen that the classifier considers this new instance for future learning and as a result eventually, the learning of the classifier can be deviated from the normal value.

5. REFERRED MODEL

The literature survey demonstrates numerous contributions on using machine learning techniques for successful intrusion detection. Some of the latest work can be found in [42, 43, 44, 45]. In our first work, we have adopted a section of the model proposed in [46]. The authors in the paper have proposed a novel framework for fully unsupervised training and online anomaly detection. Initially a model is constructed and eventually the model evolves with the status of online data. Fig. 2 shows the overview of the proposed model. The framework consists of three phases. The first phase consists of training the classification algorithm. In this phase the weight vector of a synaptic connection is adjusted by injecting the training set as input.

Once there is a winning neuron, the corresponding weight of the neuron and its neighbors defined by a neighborhood function is updated. In the second phase, the weight vector of the matured SOM is clustered, and the centroid of an attack cluster is updated resulting in change in the boundary of the clusters. In the final phase, the normal is further split into a new attack cluster. The three phases are described below.

Phase 1: Remodeling the Network Structure and Size

Whenever a new instance is fed as input, the Euclidean distance of the input vector with all the weight vectors is computed. Whichever neuron has this minimum value, becomes the winning neuron.

If \[ |x - W_{BMU}| < \mu \]

where \( \mu \) is the distance threshold.

If the above situation holds, the weights of the winning neuron and its neighbors are updated as follows

\[
W_{j}(t+1) = W_{j}(t) + \eta (x - W_{j}(t))
\]

Where \( \eta \) is the learning rate and decreases monotonically with time.

The winning neuron (BMU-Best Matching Unit) if it belongs to a normal cluster, the data falls out to be normal and vice versa.

Phase 2: Updating the centroid of the attack cluster

In this phase the centroid of the attack cluster is updated if the following condition is met.

\[
\sum_{j=1}^{m} |w_{j}(t) - w_{j}(t_0)| > \theta,
\]

i.e. the sum of the difference of the weight at a given time ‘t’ and the initial time to exceeds threshold value \( \theta \) and ‘m’ is the number of units belonging to the attack cluster.

Phase 3: Splitting the normal cluster

If \( n \)th vector is represented by \( x_n \) and ‘B’ represent a Normal cluster. Let \( B_1 \) and \( B_2 \) represent the split cluster from \( B \). Let \( \mu_B \) be the centroid of the cluster ‘i’ and ‘N’ represent the recent data points that are at a distance greater than distance \( \lambda \) from \( \mu_B \). From the direction of attack clusters, if the direction of the number of data located is different and covers a portion ‘y’ of \( N \), then k-means clustering with value of \( k=2 \) is executed on the normal cluster ‘B’ when \( SS_1/SS_2 > \beta \).

Here \( SS_1 = \sum_{x \in B} |x - \mu_B|^2 \) and \( SS_2 = \sum_{x \in B_1} |x - \mu_{B1}|^2 + \sum_{x \in B_2} |x - \mu_{B2}|^2 \)

The results after implementation of the said model were promising and is shown in the below figure.

6. PROPOSED FRAMEWORK

Adopting as inspiration the model referred in section V, the proposed model of implementation is shown below. The proposed work is divided into the following phases: (i) Preprocessing the dataset (ii) Developing the training model and (iii) Poisoning the learned model.

(i) Preprocessing the dataset.

The dataset adopted for training and testing is NSL-KDD. NSL-KDD have following advantage over KDD dataset

a) Due to absence of redundant item in the dataset, the learning does not become biased.

b) The number of selected records of each type of attack is proportional to the number of records in KDD’99.
In the first phase the dataset is preprocessed and made ready for training the learning model namely SOM & SVM. When the training set is ready, the learning model is adopted in the second phase and is trained by using the training set. Once the learning is matured, then it is tested with poison instances in the third phase. The proposed work flow of training the models is shown in the Figure 4. NSL-KDD dataset have several non-numeric attribute values. Non-numeric data cannot be adopted for training the adopted learning models. Therefore, the non-numeric data is first transformed into numeric representation and the dataset is made ready for training. Random number of lines from the KDD dataset is adopted as part of the training set. The column attributes are normalized and mapped into the interval [0,1] using min-max normalization approach. SOM is used in numerical value and in the same range. The equation for min-max normalization used is

\[ Z = \frac{x - \text{max}(x)}{\text{max}(x) - \text{min}(x)} \]

(ii) Developing the Training Model

The proposed algorithm for training the model is shown in Figure 6. The corresponding flow chart representation is shown in Figure 4. As shown in Algorithm, the input is the training set and the output is the learned model. Every instance from the training set is retrieved, preprocessed and later becomes a part of final training set. Once the training set is ready, either of the learning model can be adopted for training. If the learning model adopted is SOM, a grid of size 20x20 units is created and the units are initialized with random weight values. For every winning unit, the corresponding weight is updated as shown in the Algorithm. The above process continuous until the map is converged. Whereas, if the learning model is SVM, a kernel function is selected for training the model. In Fig 6 the linear kernel approach is shown. In such approach the objective is to find the linear hyperplane such that the support vectors of both the class are maximally separated out from each other.

(iii) Poisoning the learning model

The proposed algorithm for poisoning the learning model is shown in Fig 7. The corresponding flow chart representation is shown in Figure 5. Scapy is used to build custom packets and these packets are injected into the real network traffic. The IDS sensor running in the network captures these packets for further processing. The feature vector of each packet is extracted and fed to the classification algorithm. If the feature vector of the extracted packet is classified as ‘Normal’, the feature is added to the existing training set and becomes part of future training. If it is classified as an attack it is discarded.

The attribute values of anomaly instances in NSL-KDD is observed and packets are framed accordingly. Most of the other attributes value resembles that of normal feature set. This is done to observe the change in behavior of the classification process and variance in the detection rate and other parameters. In Fig 7, w is the set of instances. Every instance from w is preprocessed and added to the training set T until T is ready. Once T is ready, the learning algorithm is chosen in step 5. T_m is the final trained model. The attacker crafts a packet T_a and injects it into the network. If T_m is classified as normal, it becomes part of future training set T.

Game theory formulation: To ensure a high secure behavior in machine learning based IDS, the learning algorithm and its classification behavior can be portrayed as a game between the attacker and the defender. Let the attacker’s interest of corrupted training and evaluated data be A_{train} and A_{eval}.

![Fig 4: Training the learning model](image_url)

![Fig 5: Proposed flow chart for poison learning](image_url)

The game can be formulated as follows:

1. Defender: Select a learning algorithm H that can be observed as best against the observed data.
2. Attacker: Generate compromised $A^{train}$ and $A^{eval}$.

3. For learning:
   a) Receive dataset $D^{train}$ with contamination from $A^{train}$.
   b) Learn Hypothesis $f \leftarrow D^{train}$

4. Evaluation:
   a) Receive dataset $D^{eval}$ for evaluation of ‘$f$’ with or without any contamination $A^{eval}$.
   b) If the classification error rate is less than threshold accept $D^{eval}$ and may be considered for future training.

---

Training the learning model

<table>
<thead>
<tr>
<th>Input: Training set</th>
<th>Output: Learned model</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D \leftarrow Dataset$</td>
<td>$T \leftarrow Training set$, $w \leftarrow Instances$</td>
</tr>
<tr>
<td>for $w_i \in D$ do</td>
<td></td>
</tr>
<tr>
<td>$T \leftarrow preprocess w_i$ and make part of training set</td>
<td></td>
</tr>
<tr>
<td>if $T$ is ready then</td>
<td></td>
</tr>
<tr>
<td>Choose learning algorithm for training, either SVM or SOM</td>
<td></td>
</tr>
<tr>
<td>if Learning is SOM then</td>
<td></td>
</tr>
<tr>
<td>Initialize a SOM grid of size 20x20 units;</td>
<td></td>
</tr>
<tr>
<td>Initialized with random weight in the synaptic connection of the SOM grid;</td>
<td></td>
</tr>
<tr>
<td>For each input instance, identify the winning unit using Euclidean distance;</td>
<td></td>
</tr>
<tr>
<td>For every winning unit update the weight as</td>
<td></td>
</tr>
<tr>
<td>$\Delta w_i(t) = \eta(t)[y(t,k) - w_i(t)]$</td>
<td></td>
</tr>
<tr>
<td>Continue the above until the map converges</td>
<td></td>
</tr>
<tr>
<td>else if Learning is SVM then</td>
<td></td>
</tr>
<tr>
<td>$D = { (x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_n, y_n) }$</td>
<td></td>
</tr>
<tr>
<td>Where, $x_i = {x_{i1}, x_{i2}, x_{i3}, \ldots, x_{in}}$, $y_i \in {1, -1}$</td>
<td></td>
</tr>
<tr>
<td>SVM is built on a linear function</td>
<td></td>
</tr>
<tr>
<td>$f(x) = \langle w, x \rangle + b$</td>
<td></td>
</tr>
<tr>
<td>Where, $w = {w_1, w_2, w_3, \ldots, w_n}$, $b$ is bias, $\langle w, x \rangle$ is a dot product of $w$ and $x$</td>
<td></td>
</tr>
<tr>
<td>if $f(x_i) &gt; 0$ then</td>
<td></td>
</tr>
<tr>
<td>$\chi_i$ is positive</td>
<td></td>
</tr>
<tr>
<td>else if $f(x_i) &lt; 0$ then</td>
<td></td>
</tr>
<tr>
<td>$\chi_i$ is negative</td>
<td></td>
</tr>
<tr>
<td>if $\chi_i &gt; 0$ and $b &gt; 0$ then</td>
<td></td>
</tr>
<tr>
<td>$y_i = 1$</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td></td>
</tr>
<tr>
<td>$y_i = -1$</td>
<td></td>
</tr>
<tr>
<td>$f(x_1, x_2, x_3, \ldots, x_n) = \langle w_1, x_1 \rangle + \langle w_2, x_2 \rangle + \langle w_3, x_3 \rangle + \cdots + \langle w_n, x_n \rangle + b$</td>
<td></td>
</tr>
<tr>
<td>SVM is fitted a hyperplane such that $\langle w, x \rangle + b = 0$</td>
<td></td>
</tr>
</tbody>
</table>

---

7. EXPERIMENTAL RESULTS AND ANALYSIS

The different languages and packages used for implementation are as follows: Python version 2 & 3, Scikit python package and Ubuntu 14.

The experimental approach is divided into the following phases:
(a) Train SOM and SVM and test the classification result.
(b) Poison SOM and SVM with crafted instances and observe the variance in the result from the first phase

The experiment was carried out in a LAN framework as shown in Figure 8. In Figure 8, the IDS sensor is the system running machine learning based IDS software. The attacker is assumed to get hold of host pc0 and pc1. The maliciously crafted packets are injected from pc0 and pc1 into the real time traffic of the network. In the first phase of the experiment, a SOM grid of size 20x20 is initialized and trained on NSL-KDD dataset until the SOM grid is converged. For every input unit the BMU (Best Matching Unit) is recorded.

---

Fig 7: Proposed method for poisoning online learning

These BMU’s are later clustered into 20 different clusters which universally is mapped into either a normal or an attack cluster. Fig 9 shows the visual plane of weight vectors after being trained with NSL KDD Data set. Different colours of the weight vectors indicate the different clusters to which they fall. This output is on Normal Training data i.e. before subjecting to poison learning. The proposed flow chart to fail the model is portrayed in Fig 5. As seen in the proposed model poison instances are crafted by exhibiting the property “camouflage” i.e. normal instances vectors are picked up and their attributes values are varied in accordance with the value set of attack vectors.

---

Fig 8: Experimental set up

The set of attributes that attacker picks up and can influence externally are shown in Figure 10. Once the attacker crafts packet instance that seemingly looks normal but eventually in the long run may lead to a poison attack. These packets are injected into the IDS sensor. It was observed that the IDS sensor classified these instances as normal and therefore, makes them part of future training set.
The attacker exploits this behavior and gradually mislead the learning towards misclassification of true instances. One example of tampered attribute is such as Column 26 of NSL-KDD - `error_rate` (% of connections that have `SYN` errors to the same host). Table 1 illustrates the result of a normal SOM on NSL-KDD dataset. The accuracy of the detection is 85%. It is important to note here that our objective is not to improve on the accuracy but to observe if this accuracy value could be influenced by poison learning. Fig 9 shows the orientation of the BMU in SOM grid. Initially, the SOM is influenced by changing one random attribute from Fig 10.

Table 1 shows the result of training the SOM in normal circumstances. Normal circumstances here imply that the training instances are non-tampered i.e. the feature vector set used for training belongs to true normal and attack instances. The size of the SOM grid is 20x20 units and as stated earlier the weights are assigned randomly until the SOM grid is converged with training instances. The testing instances are than fed to the SOM grid. An output unit in the SOM grid claims responsibility of the input instances and therefore becomes the winning unit i.e. BMU (Best Matching Unit). In our experiment the weight vectors connecting the input unit to the output units of the SOM grid are clustered into twenty numbers after the training phase. Each of these clusters either falls into attack or normal cluster. The category of the cluster is determined by the supervised label of the training instances. A BMU corresponding a training instance marked attack is part of the attack cluster.

Table 1: Implementation results of normal SOM

| Number of training instances | 3000 |
| Execution time with mentioned hardware and software details | 35 hours |
| Total cluster into which weight vectors of SOM is clustered | 20 |
| Cluster indices that are part of attack. Each cluster consists a set of weight vectors of the SOM grid. | [0,2,5,5,7,8,10,11,12,13,14,15,16,17,18,19] |
| Cluster indices that are part of normal. Each cluster consists a set of weight vectors of the SOM grid. | [9,4,6,12] |
| Detection Rate (attack instances) | 85% |
| Precision | 77% |
| Sensitivity | 85% |
| Specificity | 67% |

From Table 1 the total number of clusters that falls in generic attack clusters is 16 and that falls in generic normal cluster is 4. The converged SOM is than tested with the training instances. With the standard testing test of NSL-KDD dataset, the detection accuracy as shown in Table 1 is 85%. However, we
would like to restate that the objective of the work in not to improve detection accuracy but to discover if a learning-based IDS can be influenced externally. With this objective packet were framed that seemed normal but eventually in the long run may lead to an attack. Attributes whose value can be influenced externally are already mentioned in Figure 10.

Table 2: Implementation results after one attribute poison

<table>
<thead>
<tr>
<th>Number of training instances</th>
<th>3000 + 1500(poison)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time with mentioned hardware and software details</td>
<td>35 hours</td>
</tr>
<tr>
<td>Total cluster into which weight vectors of SOM is clustered</td>
<td>20</td>
</tr>
<tr>
<td>Cluster indices that are part of attack. Each cluster consists a set of weight vectors of the SOM grid.</td>
<td>[1,2,4,7,10,12,14]</td>
</tr>
<tr>
<td>Cluster indices that are part of normal. Each cluster consists a set of weight vectors of the SOM grid.</td>
<td>[0,3,4,5,6,8,9,11,13]</td>
</tr>
<tr>
<td>Detection Rate (attack instances)</td>
<td>83%</td>
</tr>
<tr>
<td>False Positive Rate</td>
<td>28%</td>
</tr>
<tr>
<td>Precision</td>
<td>78%</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>83%</td>
</tr>
<tr>
<td>Specificity</td>
<td>71%</td>
</tr>
</tbody>
</table>

Table 3: Implementation results after four attribute poison learning [attack vector attributes with normal value set].

<table>
<thead>
<tr>
<th>Number of training instances</th>
<th>3000 + 1500(poison)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time with mentioned hardware and software details</td>
<td>34 hours</td>
</tr>
<tr>
<td>Total cluster into which weight vectors of SOM is clustered</td>
<td>20</td>
</tr>
<tr>
<td>Cluster indices that are part of attack. Each cluster consists a set of weight vectors of the SOM grid.</td>
<td>[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]</td>
</tr>
<tr>
<td>Cluster indices that are part of normal. Each cluster consists a set of weight vectors of the SOM grid.</td>
<td>[2,18,15]</td>
</tr>
<tr>
<td>Detection Rate (attack instances)</td>
<td>92%</td>
</tr>
<tr>
<td>False Positive Rate</td>
<td>83%</td>
</tr>
</tbody>
</table>

Table 4: Crafted packets are classified as normal by the learned IDS as result portrays no BMU falls in the Attack Cluster.

<table>
<thead>
<tr>
<th>Number of training instances</th>
<th>1500</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total cluster into which weight vectors of SOM is clustered</td>
<td>20</td>
</tr>
<tr>
<td>Cluster indices that are part of attack. Each cluster consists a set of weight vectors of the SOM grid.</td>
<td>[]</td>
</tr>
<tr>
<td>Cluster indices that are part of normal. Each cluster consists a set of weight vectors of the SOM grid.</td>
<td>[0,2,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]</td>
</tr>
<tr>
<td>Detection Rate (attack instances)</td>
<td>100%</td>
</tr>
<tr>
<td>False Positive Rate</td>
<td>0%</td>
</tr>
<tr>
<td>Precision</td>
<td>100%</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>100%</td>
</tr>
<tr>
<td>Specificity</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 2 demonstrates the result after injecting the IDS with 1500 poison instances, i.e. attributes values are modified in such manner that the IDS classify them initially as normal and eventually these instances become part of future training by the learning algorithm. It is observed that there have been altogether reorientation of the weight vectors falling into normal and attack clusters. The accuracy results have dropped from 85% to 83% as found from the experiment. This indicates that an attacker can externally influence an online learning and thereby bring the future classification result of an online IDS down. Table 3 displays the result of similar experiment repeated but with higher number of tampered attributes values. Table 4 demonstrates the result of the classification by the IDS of the instances that are programmatically crafted that seemingly are normal but are poison instances. When these instances are injected to the IDS for classification, it is observed that the clusters of BMU falling in the generic attack cluster is empty and therefore all the instances are treated normal and therefore, becomes part of future training. The detection rate is 100% indicating all the crafted instances are very well recognized as normal by the detection engine of the IDS. Citing as an example one attribute value of crafted instances that was incrementally changed was dst_host_host_count: Number of connections from the same host to the destination in the past 2 seconds.

Table 5: Classification result of a normal SVM

<table>
<thead>
<tr>
<th>Number of training instances</th>
<th>10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection rate (attack instances)</td>
<td>100%</td>
</tr>
<tr>
<td>Precision</td>
<td>100%</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>100%</td>
</tr>
</tbody>
</table>

We kept all other feature values (as per NSL-KDD) of a packet same as that of a normal packet but kept slowly rising in linear pattern the value of the above attribute. It is later
observed that the IDS eventually started to fail recognizing DoS (Denial of Service) attack in form of SYN flood performed from a single machine to a target destination. The IDS started classifying all of them eventually as normal packets. This signifies that an attacker can plan very carefully to bypass detection of a specific attack by an online IDS. Apart from testing this behavior with online based IDS using SOM as the classification tool, we also tested it with SVM (Support Vector Machine). Support Vector Machines have proven effective in classification of high dimensional data with significantly bigger training instances and attributes. SVM is trained with training set from NSL-KDD Dataset. The implementation of SVM on training samples exhibits high accuracy i.e. the SVM perfectly classifies the training and the testing instances. Ten thousand samples from NSL-KDD dataset were adopted for training the SVM. Table 5 summarizes the result of the output of the SVM. The learned SVM is tested on the NSL-KDD testing set. As seen from Table 5, with zero false positive or false negative the detection comes to 100%.

Figure 7 shows the support vectors plotted in a normal SVM trained on NSL-KDD dataset using linear kernel.

![Support vectors in a normal SVM using linear kernel](image)

It is observed from Figure 7 that none of the support vectors are misclassified. Therefore, the detection rate is high. Different colours of the panel represents instances falling to different clusters. The support vectors are labeled in the figure. Fig 7 shows the SVM plot with a linear kernel.

![Support vectors in a normal SVM using polynomial kernel](image)

Figure 8 shows the support vectors plotted using a polynomial kernel and Figure 9 shows the support vectors plotted using a radial basis function. It has been observed in all the SVM plotted figures that none of the testing instances are misclassified and the detection rate really goes well because of large size in the feature set as can be seen from Table 5. However, when the SVM is trained using poison instances as discussed before, the support vector changes as shown in Figure 10 from that of support vectors shown in Figure 7. The accuracy of detection rate drops below 100%. This is vivid by the number of misclassified support vectors as can be seen from Figure 10. In normal SVM as seen in Figure 7, there were no misclassified support vectors and therefore high detection accuracy.

![Support vectors in SVM learned using poison (manually crafted) instances using linear kernel](image)

Similarly, the misclassification in SVM using polynomial kernel can be seen in Figure 11 as that from Figure 8. Likewise, misclassification error of support vector in SVM using radial basis function can be observed in Figure 12 from...
that of Figure 9. As can be seen from Table 6, the support vectors either falls in one of the class i.e. in generic Attack or Normal. As can be seen from the table two number of support vectors falls in the first class and six number of support vectors falls in the second class. As described earlier, the framed instances are crafted keeping resemblance with the attack set vectors of NSL KDD set. However, significant changes in indices of support vector set compared to support vectors in normal SOM is observed.

This indicates that the behavior of the learning can be influenced by carefully crafting packets that may seem normal but can be a potential attack in the long run. The number of support vectors belonging to a given class also changes significantly.

Table 7: Support vector set in one attribute poisoned trained with linear kernel based SVM

<table>
<thead>
<tr>
<th>(TP, 0, TN, 500, FP, 0, FN, 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support vector class - [7(first class), 1(second class)]</td>
</tr>
<tr>
<td>Support vector indices set -- [128 292 356 419 787 885 0]</td>
</tr>
</tbody>
</table>

Table 8: Support vector set in four attributes poisoned trained with linear kernel based SVM

<table>
<thead>
<tr>
<th>(TP, 0, TN, 500, FP, 0, FN, 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support vector class - [7(first class), 1(second class)]</td>
</tr>
<tr>
<td>Support vector indices set -- [100 113 179 216 390 481 605 610 0]</td>
</tr>
</tbody>
</table>

8. MATHEMATICAL FRAMEWORK

The mathematical formulation portraying the deviation in the learning with newly injected normal and poison packets can be derived as below:

Y - inclusion rate of learning instances for normal learning; L - unaffected Learning, $\alpha$ - infectivity rate on learning by malicious instances; $X$ - set of previous malicious instances (if any) already part of the learning set; $\beta$ - error rate in the non-tampered learning; The rate of change in the learning (gradual inclination towards poison learning) can be formulated as
follows: $dL/dt = Y - aLX - \beta L$. The following equation indicates how much influence the instances that actually "attack" mode but are classified as normal and became part of future learning set that can further influence the learning: 
$dE/dt = aLX - (\lambda + 0)E$

9. CLASS IMBALANCE IN TRAINING SET

Most of the machine learning algorithms are subjected to imbalance problem [55,56]. There have been work to address the imbalance problem by different researchers [57,58]. The experiment and evaluation demonstrated in this paper is not in relation to class imbalance problem during the training. The training data generated in the experimental evaluation is free of class imbalance problem. While generating the training set almost an approximate equal number of labelled instances from each of attack and normal set were considered. It was also done in keeping in mind not to make the learning algorithm victim of overfitting problem. To ensure the same Tomek links [51] was considered. Therefore, no two examples were considered that formed Tomek links.

10. PROPOSED SOLUTION TO OVERCOME THE OBSERVED PROBLEM

Training data manipulation: From the experimental evaluation it is observed that the anomaly in the true classification is due to incorporation of instances in the future learning set that are otherwise classified as normal but may lead to poison learning in the long run. Whenever, an incoming instance is classified as normal rather than embedding this instance immediately as a part of future training set, these instances are made part of a temporary set. When the size of this temporary set is large the instances of the set are made part of the training set and the learning is made to reoccur again on this training set. Once the learning is converged, the learning algorithm is run on randomly picked samples from testing set of NSL-KDD dataset. If the detection rate drops below compared to the rate learned on the training set, then the training set remains same as the old training set i.e.

If detection_rate_{new} < detection_rate_{old}:

training_set_new = training_set_old;

Else:

training_set_new(future training set)=
training_set_old + temporary_set;

Certain methods such as RONI [52] have been proposed in certain context such as spam classification of emails in relevance to training data manipulation. However, in this aspect RONI approach might fail or prove computationally more intensive. The above proposed idea of temporary set approach would prove effective and less computationally intensive as the learning would not be invoked with every new instance. However, the degree of such efficiency would be considered in the future study and experimental evaluation.

11. CONCLUSION

The above experiments demonstrate that it is possible to influence the classification behaviour of an online based IDS by systematically changing certain attribute values of a packet feature set. Experimental evaluation shows that the detection accuracy of the online IDS declines after subjected to poison packet attacks. The experimental evaluation is significant in the sense that it gives an understanding of the necessary steps to be adopted for online learning-based IDS for safe and secure learning. It can be therefore concluded that machine learning algorithms are never blindly secure and leave a scope for analysis of such algorithms under different circumstances [47]. If the attacker has some idea of the attributes used for training purpose, he can play around with self-crafted instances with different values for those attributes for deviating the classification behavior of the learning algorithm. This work further motivates to pick up the responsive behavior of a Network subject to attack. One of such work undertaken can be found in [48]. It is also observed that people have tried to devise a different approach to achieve security at different times [49, 50]. Therefore, there always exist an enthusiasm among security researcher to design IDS/IPS or responsive system that can ensure minimum casualty to the network and organization. The experimental evaluation leaves another scope of designing a biospired response system of a network to withstand unseen attacks.

REFERENCES

Accented Manuscript


[34] Xiao, Liyuan, Yetian Chen, and Carl K. Chang. "Bayesian model averaging of bayesian network classifiers for intrusion detection." Computer Software and Applications


