International Journal of Computational Intelligence and Applications


**ACCEPTED MANUSCRIPT**

# INTELLIGENT INTRUSION DETECTION SYSTEM THROUGH COMBINED AND OPTIMIZED MACHINE LEARNING

SYED ALI RAZA SHAH and BIJU ISSAC*

*School of Computing, Media and the Arts, Teesside University*
*Middlesbrough, England, UK*
*ali.aliraza@gmail.com and bissac@ieee.org*

SEIBU MARY JACOB

*School of Engineering, Science and Design, Teesside University*
*Middlesbrough, England, UK*
*s.jacob@tees.ac.uk*

Received 27 June 2017
Revised 17 March 2018

In this paper an existing rule-based intrusion detection system (IDS) named Snort is made more intelligent through the application of machine learning. We chose Snort as it is an open source software and though it was performing well, there was the issue of false positives. To find the best performing algorithm to use on Snort to improve its detection we tested some machine learning algorithms on three available datasets. Support Vector Machine (SVM) was chosen along with Fuzzy Logic and Decision Tree based on their accuracy. Combined versions of algorithms through ensemble SVM along with other variants were tried on the generated traffic of normal and malicious packets at 10 Gbps. Optimized versions of the SVM along with Firefly and ACO were also tried, and the accuracy improved remarkably. Thus, the application of combined and optimized machine learning algorithms to Snort at 10 Gbps worked quite well.

*Keywords*: Snort Intrusion Detection, Machine Learning, Support Vector Machine, Fuzzy Logic, ACO, Firefly

## 1. Introduction

The Snort Intrusion Detection System (IDS) which is a de-facto standard currently started its development in 1998 by Sourcefire. Snort has a single threaded architecture and uses the TCP/IP stack to capture and inspect network packet payload. Snort has added a multi-instance feature to its 2.9 release to address the limitation of single-thread and the version 3.0 will be multithreaded by default (Snort, 2017). The Snort IDS is dependent on the rule set to detect malicious traffic that gives an accurate description of the known malicious traffic. To reduce the false positive rate (FPR), many researchers have used machine-

* Corresponding author

learning algorithms to classify normal and malicious traffic. Some of the machine learning algorithms that have been extensively used are Support Vector Machine, Neural Network, Decision Trees, Random Forest, Fuzzy Logic, BayesNet, NaiveBayes etc. The best-performing algorithm has to be selected to improve the detection accuracy by reducing the false positive alarms.

Before using machine-learning techniques in Snort, it is important to know about high performing machine learning algorithms. Three publicly available datasets were used to conduct performance experiments on Machine Learning Algorithms (MLAs). The Weka experimental setup and data preprocessing using data mining software is used (Weka, 2017). False positive, false negative and true positive alarm rates were the metrics used for the comparison of detection accuracy of the selected MLAs.

This paper is organized as follows. Section 2 is the related works to investigate what others have done in the topic being researched, followed by section 3 on the architecture of Snort. Section 4 is modeling the attack through Kali Linux, section 5 on the analysis of machine learning algorithms on datasets, followed by section 6 on a discussion of SVM and ensemble SVM. Section 7 is on intelligent plug-in development that discussed on single, combined and optimized algorithms, followed by section 8 which is the conclusion.

## 2. Related Works

Yin et al. (2017) proposed a deep learning approach for intrusion detection using recurrent neural networks (RNN-IDS). The experimental results show that RNN-IDS is very suitable for modeling a classification model with high accuracy and that its performance is superior to that of traditional machine learning classification methods in both binary and multiclass classification. Zhang and Zhu (2018) proposed a privacy-preserving machine-learning-based collaborative IDS (PML-CIDS) for Vehicular ad hoc network (VANETs). The proposed algorithm employs the alternating direction method of multipliers to a class of empirical risk minimization problems and trains a classifier to detect the intrusions in the VANETs. The authors use the differential privacy to capture the privacy notation of the PML-CIDS and propose a method of dual-variable perturbation to provide dynamic differential privacy. They used NSL-KDD dataset to corroborate the results on the detection accuracy. Massato Kakihata et al. (2017) did an analysis on network flows previously collected and correctly detected three different types of attacks. The flows were organized to be processed by machine learning methods, getting promising results. Al-Jarrah et al. (2016) proposed a state-of-the-art Traffic-based IDS (T-IDS) built on a novel randomized data partitioned learning model (RDPLM), relying on a compact network feature set and feature selection techniques, simplified subspacing and a multiple randomized meta-learning technique. The proposed model has achieved 99.984% accuracy on a well-known benchmark botnet dataset.

Wang et al. (2016) proposed a new hybrid learning method based on features such as density, cluster centers, and nearest neighbors (DCNN). Data is represented by the local density of each sample point and the sum of distances from each sample point to cluster centers and to its nearest neighbor. k-NN classifier is adopted to classify the new feature vectors which is effective in intrusion detection. Kolias et al. (2016) categorized and thoroughly evaluates the most popular attacks on 802.11 and analyzed their signatures. They analysed a publicly available dataset containing a rich blend of normal and attack traffic against 802.11 networks. An extensive first-hand evaluation of this dataset using several machine learning algorithms and data features is also provided. Hu et al. (2008) proposed an intrusion detection algorithm based on the AdaBoost algorithm where decision stumps are used as weak classifiers. The decision rules are provided for both categorical and continuous features. Adaptable initial weights and a simple strategy for avoiding overfitting are adopted to improve the performance of the algorithm. Experimental results show that our algorithm has low computational complexity and error rates.

## 3. The Architecture of Snort

There are four basic components to the architecture of Snort. They are the packet sniffer, the preprocessor, the detection engine and the output. Snort will receive packets and process them through preprocessor and compare these packets against the set of rules. The output will log or trigger alerts based on what action the rules will take.

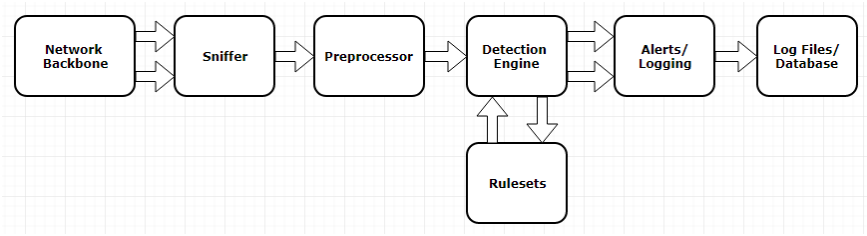


Fig 1. The Snort Architecture

The default rule set was used in Snort. Using Kali Linux Metasploit framework the seven types of malicious traffic were generated as shown in Table 1, along with the legitimate traffic. They were all injected to Snort to simulate the attacks. The IDS will inspect the legitimate and malicious traffic and will trigger alarms when the input traffic matches the rule set. The number of common rule set used is shown in Table 1.

The table 2 shows the detection accuracy of Snort when it was tested. There was a high percentage of FPR as 56.2% and 6.0% as FNR.

Table 1. Number of common rule set

| No | Rules and Malicious Traffic Type | Number of Rules |
|---|---|---|
| 1 | SSH | 13 |
| 2 | DoS/DDoS | 69 |
| 3 | FTP | 75 |
| 4 | HTTP | 110 |
| 5 | ICMP | 125 |
| 6 | ARP | 21 |
| 7 | SCAN | 30 |

Table 2. Malicious traffic accuracy (%) measurements at 10 Gbps for 10 hours

| Malicious Traffic | Snort FPR | Snort FNR |
|---|---|---|
| SSH | 9.3 | 0.0 |
| DoS/DDoS | 3.3 | 0.8 |
| FTP | 9.6 | 0.0 |
| HTTP | 6.3 | 1.1 |
| ICMP | 16.9 | 1.0 |
| ARP | 7.9 | 0.9 |
| Scan | 2.9 | 2.2 |
| Total | 56.2 | 6.0 |

## 4. Modeling the Attack through Kali Linux

This experiment required a target server running HTTP, FTP and SSH services as shown in Figure 2 and the software configured in them are shown in table 3.

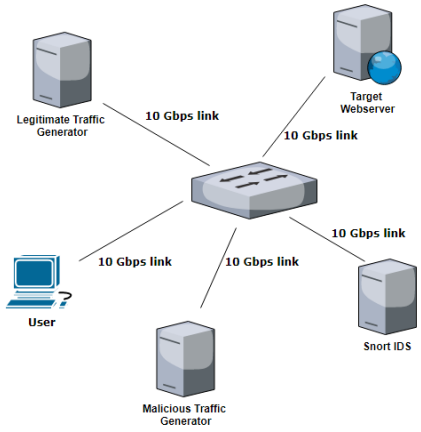


Figure 2. The Experiment Network

Table 3. Experiment network specifications

| **Machine/Software Type** | **Specification** | **Tools Used** |
|---|---|---|
| CentOS 6.5 | Virtual Machine, 2.5 GHz 4 cores CPU, 4 GB Memory, 10 Gbps Ethernet | Snort 2.9.6.1 IDS; Collectl, top, dstat Snort logs, tcpdump, IPTRAF |
| Malicious Network Traffic Generator | Virtual Machine, 2 cores CPU, 2 GB Memory, 10 Gbps Ethernet | Kali Linux Metasploit Framework |
| User PC | Virtual Machine, 2.5 GHz 4 cores CPU, 4 GB Memory, 10 Gbps Ethernet | Ubuntu client |
| Legitimate Network Traffic Generator | Virtual Machine, 2 cores CPU, 2 GB Memory, 10 Gbps Ethernet | Ostinato, NMAP, NPING |
| Network Switch | Virtual Switch | |

The normal network traffic for the experiments was produced through three open source network traffic generators, namely Ostinato, NMAP, and NPING (Heikur, 2015). These tools can generate network traffic up to 20 Gbps. The Kali Linux Metasploit framework generated seven types of malicious traffic as in Table 1, along with the legitimate traffic (Agarwal and Singh, 2013). The IDS will inspect the traffic and will trigger alarms when the input traffic matches the rule set. The number of instances of alarms like false positive, false negative and true positive will show the accuracy of Snort's detection capability. The Metasploit framework generates malicious traffic with different exploits and payloads and provides different exploits and payloads for different operating systems. For example, to successfully execute the payload, the victim computer should be found to have an exploit with open ports through a scan. The basic steps for exploiting a system are as follows:

1. Select and configure an exploit which will create an advantage through the security weaknesses and bugs in an operating system.
2. Select and set up a payload that will be the executable program that will run on the target machine.
3. Select an encoding technique that prevent detection by the IDS.
4. Run the exploit.

During an attack, to run an exploit the information on the attack target system is needed like the type of operating system, port information etc. Scanning and fingerprinting tools such as Nmap, can collect this data. Since the Metasploit framework is flexible and modular, different payloads and exploits can be tried to achieve the best outcome. The example of Snort IDS rule is as follows.

- alert icmp any any → any any (msg:"ICMP Packet"; sid:1000967; rev:1;). This rule does alerting when there is an ICMP packet (ping traffic). The *sid* (signature id) keyword is used to uniquely identify Snort rules. The *rev* keyword is used to uniquely identify revisions of Snort rules.
- alert tcp any any → any any (msg: "Sample alert";). This rule does alerting when there is an TCP traffic.

The architecture of Metasploit uses libraries, especially the Ruby Extension (Rex) library. These libraries consist of a collection of tasks, functions and operations used by the framework. Rex has no dependencies and has the exploitation utility class along with others. The MSF core library extends Rex and allows to communicate with exploit modules. The core library extends the base library which is connected to a different user interface that helps to use command console and the web interface. The later version of the Metasploit framework has around 1412 exploits, 802 auxiliaries, 361 payloads, 327 encoders and 8 nops. An exploit is the entry point that the attacker takes advantage of in a system, service or application. A payload is the program that can be executed on the system that is exploited. The auxiliary module gives additional options for fuzzing, scanning, recon, DoS attack etc. The encoders are used to obfuscate modules to masquerade and avoid detection by security systems like antivirus or firewall. Nop is No Operation that prevents the payload from crashing while executing jump statements in the shell code (Agarwal and Singh, 2013).

Some examples of attack patterns are as follows. Through a network scan or by wireless packet capture, the attacker can get enough information to execute an exploit remotely. He can allow an attractive software with a hidden program to be downloaded free of charge, where the spy program could run in the background creating a backdoor entry. He can use some stolen information through social engineering to do a server-side or client-side attack.

The access privileges of the attacker can thus be escalated and that could lead to accessing and stealing of data or exploration of the system. Then there is also the process of pivoting that allows the attacker to move to other services through exploiting the initial vulnerability. The administrator can be alerted by an intrusion detection system when the attacker goes into the next stage after the network scan. The common performance measurement parameters of the IDS detection accuracy are as follows.

True Positive (TP): True is identified as true
True Negative (TN): False is identified as false
False Positive (FP): True is identified as false
False Negative (FN): False is identified as true

P = TP + FN (number of correct identification)
N = FP + TN (number of wrong identification)

True Positive Rate (TPR) = TP / P = TP / (TP + FN) (1)
False Positive Rate (FPR) = FP / N = FP / (FP + TN) (2)
False Negative Rate (FNR) = FN / P = FN / (FN + TP) (3)

## 5. Analysis of Machine Learning Algorithms on Datasets

We had used open source machine learning software Weka that is capable of data pre-processing and classification on three public datasets, namely NSA Snort IDS Alert Logs, DARPA IDS Dataset and NSL-KDD IDS Dataset were selected and used for the experiments (Deraman et al, 2011). The details of these datasets are listed in Table 4.

Table 4. Details of the Four Datasets

| **Dataset Name** | **Attack Types** |
|---|---|
| NSA Snort IDS Alert Logs (SIDL, 2018) | • MAC Spoofing<br>• DNS Poisoning<br>• IP Spoofing |
| DARPA IDS Dataset (DARPA IDS, 2018) | • SSH Attacks<br>• FTP Attacks<br>• Scanning Attacks |
| NSL-KDD IDS Dataset (NSS-KDD, 2018) | • Denial of Service Attack (DoS)<br>• User to Root Attack (U2R)<br>• Remote to Local Attack (R2L)<br>• Probing Attack |

We used the following five algorithms for testing detection accuracy on different IDS datasets, as they performed the best with Weka.

A Support Vector Machine (SVM) is a discriminative classifier which supports both regression and classification tasks that is based on decision planes that define decision

boundaries in a multidimensional space that separates cases of different class labels. A set of objects having different class memberships is separated by a decision plane. For example, if we think of two labeled classes, the SVM finds out a line or hyperplane to separate the classes. It can handle multiple continuous and categorical variables (Cortes and Vapnik, 1995).

Decision tree is a predictive model that uses observations about an item that is denoted in the branches to predictions about the target value of the item that is denoted in the leaves. It is a model that uses several input variables to predict the value of a target variable. It is drawn upside down with its root at the top. The tree has nodes that are the features or attributes, and each link denotes a decision or rule and each leaf denotes an outcome that are categorical or continuous values (Liao et al., 2010).

Fuzzy Logic is more of a many-valued logic in which the truth-values of variables may assume a value between 0 and 1, thus representing partial truth, which falls between truth and falsehood. Traditional logic classifies information into binary patterns, for example like yes or no and true or false. Between these black and white scenarios like completely true or completely false, there are possibilities and fuzzy logic focuses on using that space in between. Fuzzy systems with fuzzy if or if-then rules are being used in different scenarios (Lowen and Roubens, 1993).

Bayesian network is a model that represents the possible states of a scenario or world and it has probability relationships between some of the states. It is a probabilistic graphical model that denotes a set of variables and their conditional dependencies through a directed acyclic graph (DAG). There are three main inference tasks for Bayesian networks, which are inferring unobserved variables through observing the evidence variables, parameter learning and structure learning (Hussain et al., 2007).

Naive Bayes classifiers are simple probabilistic classifiers that works using Bayes' theorem with the features having strong and independence assumptions between them. It is very good when the dimensionality of the inputs is high and can often outperform other sophisticated classifiers (Zhang and Su, 2008).

The five algorithms mentioned were tested on four datasets. The k-fold cross validation was done with 90% of data used for training the classifier and 10% of data used for testing it. This was repeated 10 times as a circular cycle. The activity was done again with 80% of data and 20% of data for training and testing respectively. The average values are shown in the table 5.

The metrics used were false positive rate (FPR), false negative rate (FNR) and true positive rate (TPR) of each classifier and they were obtained from Weka. The detection accuracy (DR) was calculated based on equation 4.

$$DR = TPR / (TPR + FNR) \quad (4)$$

The FPR was also noted along with DR and as per table 5, Support Vector Machine classifier showed the highest accuracy followed by Fuzzy Logic, Decision Trees, BayesNet and NaiveBayes.

Table 5. The accuracy of algorithms on the datasets

| **NSA Snort IDS Alert Logs** | | |
|---|---|---|
| Machine Learning Algorithms | DR | FPR |
| Support Vector Machine | 96.4% | 0.7% |
| Decision Trees | 78.6% | 3.3% |
| Fuzzy Logic | 93.8% | 0.7% |
| BayesNet | 64.7% | 4.3% |
| NaiveBayes | 61.3% | 4.1% |
| **DARPA IDS Dataset** | | |
| Machine Learning Algorithms | DR | FPR |
| Support Vector Machine | 96.2% | 0.9% |
| Decision Tree | 80.7% | 2.5% |
| Fuzzy Logic | 91.2% | 2.3% |
| BayesNet | 62.1% | 5.7% |
| NaiveBayes | 64.4% | 6.5% |
| **NSL-KDD IDS Dataset** | | |
| Machine Learning Algorithms | DR | FPR |
| Support Vector Machine | 97.0% | 3.5% |
| Decision Tree | 76.5% | 11.2% |
| Fuzzy Logic | 94.3% | 4.3% |
| BayesNet | 67.9% | 8.5% |
| NaiveBayes | 68.5% | 7.9% |

Snort in conjunction with machine learning algorithms can accurately classify the legitimate and malicious traffic. It also improves Snort detection accuracy and reduces the false positive rate. This is explored in the next few sections.

## 6. SVM and SVM Ensemble

Based on the results in Table 5, the SVM was selected. SVM is used in a supervised environment that can train with a large number of patterns and it is used to solve a two-outcome classification or binary classification problem. SVM segregates instances from different classes through the use of hyperplane and ensures that all the instances are outside the margin. This gives rise to a hard margin that can be shown below:

$$y_i(w.x_i + b) \geq 1 \, for \, 1 \leq i \leq n, w \in R^d, w \in R \tag{5}$$

where $x_i$ denotes instances, $y_i \in \{-1, 1\}$ are labels of instances, an intercept term is $b$, $w$ is normal vector to the hyperplane, $d$ is the dimension of input vector and $n$ is the number of input data.

The hyperplane is not easily found in a real-world scenario, as data usually have a few outliers, where within the same class the instances vary significantly from other instances. The soft margin was suggested to deal with this issue is shown below.

$$y_i(w.x_i + b) \geq 1 - \epsilon_i, \epsilon_i \geq 0, 1 \leq i \leq n \tag{6}$$

where $\epsilon_i$ denotes slack variables, which allow instances to fall off the margin. To find the optimal soft margin, the following should be noted.

$$\min \frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{n} \epsilon_i \tag{7}$$

where $C$ is the soft margin cost function, which controls the classification accuracy.

Since we are not dealing with linearly separable data, the kernel function is used to replace the dot product. The Radial basis function (RBF) which satisfies Mercer's condition is the most commonly chosen and it is defined as follows.

$$K(x_i, x_j) = \exp(-\gamma \parallel x_i - x_j \parallel^2) \tag{8}$$

The parameter $\gamma$ shows the influence of single training example reaches with low values showing far reach and high values showing close reach.

There are various methods for constructing an ensemble of classifiers. The key aspect of SVM ensemble is that individual SVM should be different from the other as much as possible and this is achieved by using different training sets for each SVMs. To select the training samples, some methods used are bagging, boosting, randomization, stacking and

dagging (Kim et al., 2003). In bagging, various SVMs are independently trained through a bootstrap method and later combined through an appropriate combination technique. Similar to the bagging method, in boosting through the use of AdaBoost algorithm (Freund and Schapire, 1997), each SVM is trained using a different training set but the selection scheme of training samples in the AdaBoost algorithm used as in figure 2 is quite different. After training, several independent SVMs are combined through linear or non-linear methods. Of the linear methods namely, majority voting and LSE-based weighting, the latter is used as per the results in the paper along with boosting (Kim et al., 2003). The LSE-based weighting considers several SVMs in the SVM ensemble with different weights which is dependent on the accuracies of classifications (Kim and Kim, 1997).

**Input:**

A set *TR* of *l* labeled examples: S = $\{(x_i; y_i)$, i=1, 2, … , $l\}$,

Labels $y_i \in$ Y = {1, 2, … , *C*}.

$p_0(x_i) := 1/l$.

**for** *k* = 1 to *K*

Build $TR_{boost_k}$= $\{(x_i; y_i)$, i=1, 2, … , $l\}$ based on the $p_{k-1}(x_i)$.

Train the *k*th SVM $h_k$ using$TR_{boost_k}$.

$\epsilon_k := \sum_{i=1}^{l} p_l(i)|\{i|h_k(x_i) \neq y_i\}|$

$\alpha_k := \frac{1}{2} ln\left(\frac{\epsilon_k}{1-\epsilon_k}\right)$.

**for** *i* = 1 to *l*

$$p_{k+1}(x_i) = \frac{p_k(x_i)}{Z_k} \times \begin{cases} \exp(-\alpha_k) \; if \; h_k(x_i) = y_i, \\ \exp(\alpha_k) \quad if \; h_k(x_i) \neq y_i. \end{cases}$$

where $Z_k$ is a normalization factor to make $\sum_{i=1}^{l} p_{k+1}(x_i) = 1$

**end**

**end**

Fig 2. The AdaBoost Algorithm (Kim et al., 2003)

## 7. The Intelligent Plugin Development

Snort has successfully detected malicious traffic as in table 2, but it triggered high false positive alarms (56.2% average). The proposed new architecture of Snort IDS shown in Figure 3 is to reduce the false positive alarms though the use of intelligent algorithms discussed before.

The intelligent plug-in will work in parallel with the Snort's rule set which detects the known malicious traffic. The plugin itself could detect new attacks patterns or malicious traffic which could reduce the false positive alarms. The existing Snort architecture shown in figure 1 is modified with an additional intelligent plug-in to produce a new architecture as shown in figure 3. The pre-processor will receive the network traffic and would feed that for intrusion detection. The proposed new architecture had the following processes:

(1) Packet Decoding**:** This process decodes the packet information to find the source and destination IP addresses, source and destination ports, source and destination Ethernet addresses, the network frame size and packet size. (2) Packet Classification: This segregates the good and bad traffic. (3) Machine Learning Algorithm**:** The intelligent plug-in uses different algorithms like SVM, Fuzzy Logic, Decision Tree, hybrid of SVM and Fuzzy Logic, Ensemble SVM, optimized SVM with firefly and ACO algorithm to process the legitimate and malicious traffic. The alarms are logged and analyzed at the end.

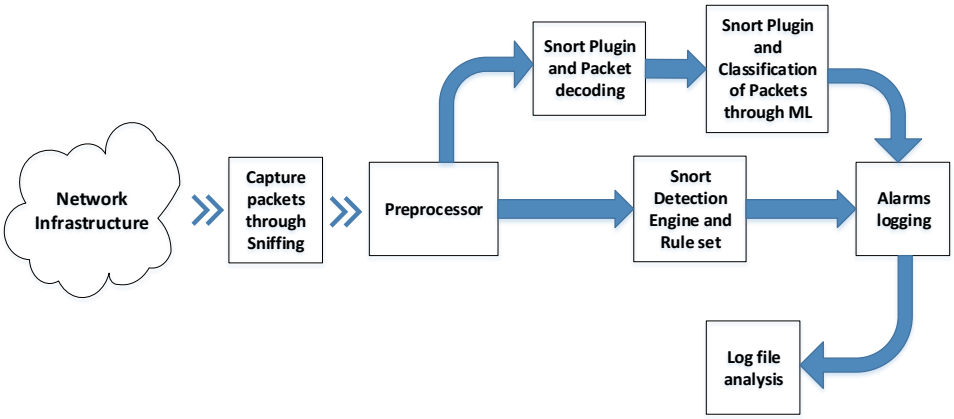


Fig 3. Proposed Snort IDS Architecture with Snort Intelligent Plug-in

The Snort intelligent plug-in for Snort v2.9 intrusion detection system was implemented in C programming language on the Linux operating system. The incoming network traffic flow is properly classified which would reduce the false positive alarms and the true positive alarms are sent to the log files of Snort's log. As per table 4, SVM and Fuzzy Logic algorithms showed good performances when tested through the Weka engine with three different datasets and Decision Tree was quite moderate in its performance. However, in order to select the superior MLA, SVM, Fuzzy Logic and Decision Tree algorithms were used along with a live background with malicious traffic to evaluate the false positive and false negative alarms rates.

### *7.1 Individual and Ensemble Algorithms*

The experiments were initially done for 10 hours in one session on the three individual basic algorithms using seven types of malicious traffic as shown in Table 1. It was repeated for SVM ensemble as well as per Stork et al., (2015). In SVM the cost function was tested with different values and the value of 1.0 was selected. The gamma values were also tested with different values to tackle under-fitting and over-fitting and a value of 0.1 was used. the Snort with SVM intelligent plug-in were injected with seven types of malicious traffic

and later the same traffic was injected into Snort with Fuzzy Logic and Decision Tree plug-in respectively. Table 6 records the FPR and FNR for four cases when they were used with Snort.

Table 6. Performance of SVM, Ensemble SVM, Fuzzy Logic, and Decision Tree plugins

| Malicious Traffic | Snort with SVM Plug-in (%) | | Snort with Ensemble SVM Plug-in (%) | | Snort with Fuzzy Logic Plug-in (%) | | Snort with Decision Tree Plug-in (%) | |
|---|---|---|---|---|---|---|---|---|
| | **FPR** | **FNR** | **FPR** | **FNR** | **FPR** | **FNR** | **FPR** | **FNR** |
| **SSH** | 3.1 | 0.1 | 2.8 | 0.0 | 4.5 | 2.1 | 9.2 | 1.9 |
| **DoS/DDoS** | 1.1 | 0.9 | 0.8 | 0.8 | 6.9 | 0.4 | 7.8 | 1.1 |
| **FTP** | 4.3 | 0.7 | 3.4 | 0.7 | 2.6 | 0.0 | 5.0 | 0.9 |
| **HTTP** | 1.8 | 1.1 | 1.6 | 1.5 | 8.0 | 1.8 | 11.8 | 0.8 |
| **ICMP** | 4.2 | 1.0 | 3.5 | 0.7 | 12.9 | 0.0 | 12.4 | 0.9 |
| **ARP** | 2.3 | 0.1 | 1.8 | 0.0 | 1.8 | 0.0 | 3.4 | 0.8 |
| **Scan** | 1.1 | 0.8 | 0.6 | 0.6 | 1.0 | 0.1 | 2.1 | 0.9 |
| **Total** | 17.9 | 4.7 | 14.5 | 4.3 | 37.7 | 4.4 | 51.7 | 7.3 |

In the initial experiment (refer to Table 2) Snort was tested without the intelligent plug-in. When seven types of malicious traffic were injected into Snort, it triggered an average value of 56.2% FPR and 6.0% FNR. When Snort was tested with ensemble SVM intelligent plug-in as in Table 6, it only triggered 14.5% FPR and 4.3% FNR, compared to SVM, Fuzzy logic and DT. This shows that the detection accuracy has really improved with SVM and ensemble versions.

### *7.2 Combined Algorithms*

The experiment was tried with a combined version of SVM with Fuzzy Logic (Karthik et al, 2016) and SVM with Decision Tree. Table 7 shows the FPR and FNR for the combined algorithms when they were used with Snort.

With the ensemble SVM and Fuzzy Logic, for each given input the basic SVM takes a set of input data and predicts two possible classes of output. That makes it a binary linear classifier that is non-probabilistic in nature. From the output of SVM, the decision-making rule of fuzzy logic is used and the results generated. The basic approach of using ensemble SVM and Decision Trees (Kumar and Gopal, 2010) is to use decision trees to approximate the decision boundary of SVM. The resulting tree is a hybrid or combined tree that has both univariate and multivariate (SVM) nodes. The tree takes SVM's help only in

classifying crucial datapoints lying near decision boundary, while the remaining less crucial datapoints are classified by fast univariate nodes. This approach focuses on reducing the number of test datapoints that need SVM's help in getting classified. With the combined version of SVM and Fuzzy logic implemented, the FNR was 12.4% and FNR was 3.1%. This is a better result than the individual ones in Table 6, though the combined version of SVM and Decision Tree did not improve the detection compared to combined Fuzzy logic approach.

Table 7. Performance of Combined Ensemble SVM and Fuzzy logic and Combined Ensemble SVM and DT plugins

| **Malicious Traffic** | **Snort with Ensemble SVM and Fuzzy Logic Combined Plug-in (%)** | | **Snort with Ensemble SVM and Decision Tree Combined Plug-in (%)** | |
|---|---|---|---|---|
| | **FPR** | **FNR** | **FPR** | **FNR** |
| **SSH** | 2.1 | 0.0 | 5.2 | 2.0 |
| **DoS/DDoS** | 0.9 | 0.4 | 7.6 | 0.5 |
| **FTP** | 3.0 | 0.4 | 2.5 | 0.5 |
| **HTTP** | 1.8 | 1.1 | 7.5 | 0.3 |
| **ICMP** | 1.6 | 0.6 | 12.5 | 0.4 |
| **ARP** | 2.1 | 0.0 | 2.5 | 0.8 |
| **Scan** | 0.9 | 0.6 | 2.0 | 0.6 |
| Total | 12.4 | 3.1 | 39.8 | 5.1 |

### *7.3 Optimized Algorithms*

The ensemble SVM classifier was optimized with firefly algorithm (Sharma et al., 2013 and Tuba et al., 2016) and it produced the best result compared to the others. SVM parameters (average value) determined by firefly algorithm are as follows: C = 1.57 and $\gamma$ = 0.58. The firefly algorithm is inspired by the flashing behaviour of fireflies and is a metaheuristic algorithm initially proposed by Xin-She Yang. Since all the fireflies are considered unisex they are not attracted based on the sex but through their brightness. The less bright firefly will move towards the brighter one, with the brightness reversely proportional to the distance.

The brightness of a firefly at a given location $x$ is given as follows.

$$I(x) = \begin{cases} \frac{1}{f(x)} & if\ f(x) > 0 \\ 1 + |f(x)| & otherwise \end{cases} \quad (9)$$

The attractiveness of a firefly ($\beta$) depends on the distance between the firefly and the one who looks at it, which is proportional to the light intensity of the firefly. It is shown as follows.

$$\beta(r) = \frac{\beta_0}{1+\gamma r^2} \quad (10)$$

The position of a firefly $i$ attracted to another brighter firefly $j$ is as follows.

$$x_i^{t+1} = x_i^t + \beta e^{\gamma r^2} ij \left(x_j^t - x_j^t\right) + \alpha_t \epsilon_i^t \quad (11)$$

where α is randomization parameter, $\epsilon_i^t$ is a vector of random numbers drawn from a Gaussian distribution or uniform distribution at time $t$, and $r_{ij}$ is distance between fireflies $i$ and $j$. The distance between fireflies $i$ and $j$ is calculated using Cartesian distance. The firefly is used to optimize the values of $C$ and $\gamma$ for SVM. Thus, the optimized parameters of SVM are calculated through firefly algorithm and is given as input to the SVM classifier. Each firefly is compared against the other and the best location is selected based on the firefly's brightness. The FPR was 9.0% and FNR was 1.7%, which is the best result we have achieved as in table 8.

The ensemble SVM was also optimized with Ant Colony Optimization (Jinyu and Xin, 2009 and Acevedo et al., 2006)) to select the parameters of SVM automatically and that proved to be effective. Ant Colony Optimization (ACO) is a metaheuristic algorithm that uses the idea exhibited in an ant colony to find the shorted path from a food source to the nest through pheromone information without employing any visual cues. When ant colony are seeking for food, they secrete a chemical substance called pheromone which act like hormones outside the body of the secreting ant. The more ants walk through that path, the more pheromone is left on the ground like trail. Then the following ant will choose one path where the probability is proportional to the amount of pheromone in that path. Finally, a shortest path from their nest to the food source is done through this positive feedback activity. To apply the colony ant behavior to ACO algorithm, the following rules were designed to make the algorithm work.

The translate rule is the transition probability among the intervals of each variable is defined as follows:

$$p_{ij} = \begin{cases} {\tau_j}^{\alpha}(\eta_{ij})^{\beta}, \eta_{ij} > 0 \\ 0, \eta_{ij} \leq 0 \end{cases} \quad i, j \in \{1,2,\dots,n\} \quad (12)$$

where $\tau_j$ denotes the pheromone intensity of the $j^{th}$ solution; $\eta_{ij}$ shows the heuristic information defined as $\eta_{ij} = f_j - f_i$; $\alpha, \beta > 0$, denotes the heuristic factors.

The transition rule is given as follows:

$$j = \begin{cases} \arg\max\{p_{ij}\}, & Translate\ to\ the\ solution\ of\ j, \\ else, & Not \end{cases} \quad (13)$$

The pheromone updating rule of the solution $j$ is expressed as follows:

$$\tau_j(t+1) = \rho\tau_j(t) + \sum_{k=1}^{m} \Delta\tau_j^k, j = 1, 2, \dots, n \quad (14)$$

$$\Delta\tau_j^k = \begin{cases} QL_j^k, L_j^k > 0 \\ 0, \quad L_j^k < 0 \end{cases} \quad j = 1, 2, \dots, n \quad (15)$$

where, $\Delta\tau_j^k$ corresponds to the increase of pheromone of the solution $j$; $L_j^k$ shows the changing value of the objective function; $\rho \in (0,1)$ denotes the pheromone maintenance; $Q$ denotes the intensity of pheromone that is the left by the ant. SVM parameters (average value) determined by the ACO are comparable to the previous effort, even though firefly optimization performed better. The FPR was 10.6% and FNR was 4.5%, which is the second best result we have achieved as in table 8. The results of both the optimized approach can be seen in table 8.

Table 8. Performance of Optimized Ensemble SVM with Firefly and Optimized Ensemble SVM with ACO plugins

| **Malicious Traffic** | **Snort with Optimized Ensemble SVM with Firefly Plug-in (%)** | | **Snort with Optimized Ensemble SVM with ACO Plug-in (%)** | |
|---|---|---|---|---|
| | **FPR** | **FNR** | **FPR** | **FNR** |
| **SSH** | 1.8 | 0.0 | 2.5 | 0.2 |
| **DoS/DDoS** | 1.1 | 0.1 | 1.5 | 0.3 |
| **FTP** | 1.9 | 0.0 | 2.2 | 0.5 |
| **HTTP** | 1.7 | 0.7 | 1.5 | 1.3 |
| **ICMP** | 1.2 | 0.6 | 1.2 | 0.9 |
| **ARP** | 0.9 | 0.1 | 1.5 | 0.5 |
| **Scan** | 0.4 | 0.2 | 0.2 | 0.8 |
| Total | 9.0 | 1.7 | 10.6 | 4.5 |

## 8. Conclusion

This paper conducted a study on the possible machine learning algorithms that can be used with Snort by first conducting an analysis through Weka on three different IDS datasets. SVM was found to perform the best, followed by Fuzzy logic and Decision Tree. The single and ensemble versions of SVM was then applied to Snort to make the detection better, followed by combined versions of SVM and fuzzy logic and SVM and Decision Tree. The results were better than the singular versions. Finally, the optimized versions of ensemble SVM with firefly and ACO is implemented and results were noted. The false positive rate

and false negative rate reduced significantly in optimized versions, especially with firefly optimization with a FPR of 9.0% and FNR of 1.7%. The results will give an insight into possible ways that one can make a rule-based IDS more intelligent.

## References


1. A. Sharma, A. Zaidi, R. Singh, S. Jain, A. Sahoo, Optimization of SVM classifier using Firefly algorithm, in: 2013 IEEE Second International Conference on Image Information Processing, Shimla, 2013, pp. 198–202.
2. C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.
3. C. Kolias, G. Kambourakis, A. Stavrou and S. Gritzalis, "Intrusion Detection in 802.11 Networks: Empirical Evaluation of Threats and a Public Dataset," in IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 184-208, Firstquarter 2016.
4. C. Yin, Y. Zhu, J. Fei and X. He, "A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks," in IEEE Access, vol. 5, pp. 21954-21961, 2017.
5. D. Kim, C. Kim, Forecasting time series with genetic fuzzy predictor ensemble, IEEE Trans. Fuzzy Systems 5 (4) (1997) 523–535.
6. DARPA IDS, DARPA intrusion detection data sets. http://www.ll.mit.edu/ideval/data/ (Accessed: January 2018).
7. E. Massato Kakihata et al., "Intrusion Detection System Based on Flows Using Machine Learning Algorithms," in IEEE Latin America Transactions, vol. 15, no. 10, pp. 1988-1993, Oct. 2017.
8. E. Tuba, L. Mrkela, M. Tuba, Support vector machine parameter tuning using firefly algorithm, in: 2016 26th International Conference Radioelektronika, Kosice, 2016, pp. 413–418.
9. F. K. Hussain, E. Chang and O. K. Hussain, "State of the art review of the existing bayesian-network based approaches to trust and reputation computation," Second International Conference on Internet Monitoring and Protection (ICIMP 2007), San Jose, CA, 2007, pp. 26-26.
10. H. Kim, S. Pang, H. Je, D. Kim, S. Y. Bang, Constructing support vector machine ensemble, Pattern Recognition, Volume 36, Issue 12, (2003) 2757-2767.
11. H. Liao, C. Alberti, M. Bacchiani, O. Siohan, Interspeech, ISCA 2010, 2958 – 2961, 2010.
12. H. Zhang & J. Su (2008) Naive Bayes for optimal ranking, Journal of Experimental & Theoretical Artificial Intelligence, 20:2, 79-93
13. J. Acevedo, S. Maldonado, S. Lafuente, H. Gomez, P. Gil (2006) Model Selection for Support Vector Machines Using Ant Colony Optimization in an Electronic Nose Application. In: Dorigo M., Gambardella L.M., Birattari M., Martinoli A., Poli R., Stützle T. (eds) Ant Colony Optimization and Swarm Intelligence. ANTS 2006. Lecture Notes in Computer Science, vol 4150. Springer, Berlin, Heidelberg.
14. J. Stork, R. Ramos, P. Koch, W. Konen (2015) SVM Ensembles Are Better When Different Kernel Types Are Combined. In: Lausen B., Krolak-Schwerdt S., Böhmer M. (eds) Data Science, Learning by Latent Structures, and Knowledge Discovery. Studies in Classification, Data Analysis, and Knowledge Organization. Springer, Berlin, Heidelberg.
15. M. A. Kumar and M. Gopal, A hybrid SVM based decision tree, Pattern Recognition, Volume 43, Issue 12, 2010, 3977-3987.
16. M. Agarwal, A. Singh, Metasploit Penetration Testing Cookbook, second ed. Packt Publishing, Birmingham, UK, 2013, pp. 1–320.
17. M. Deraman, A. Desa, and Z.A. Othman, Public domain datasets for optimizing network intrusion and machine learning approaches, in: 2011 3rd Conference on Data Mining and Optimization (DMO), 2011, pp. 51–56.

18. N. Heikura, Analyzing offensive and defensive networking tools in a laboratory environment, Tampere University of Technology, Finland, 2015 (Master of Science thesis).
19. NSL-KDD, The NSL-KDD Dataset, 2014. [Online] Available at: http://nsl.cs.unb.ca/NSL-KDD/ (Accessed: 4 February 2018).
20. O. Y. Al-Jarrah, O. Alhussein, P. D. Yoo, S. Muhaidat, K. Taha and K. Kim, "Data Randomization and Cluster-Based Partitioning for Botnet Intrusion Detection," in IEEE Transactions on Cybernetics, vol. 46, no. 8, pp. 1796-1806, Aug. 2016.
21. R. Karthik, S. Veni, B.L. Shivakumar, Fuzzy based support vector machine classifier with weiner filter system (FSVM-WF) for intrusion detection, Int. J. Adv. Res. Comput. Sci. 7 (4) (2016) 11–15
22. R. Lowen and M.R. Roubens, Fuzzy Logic: State of the Art (Springer Netherlands, 1993).
23. SIDL, Snort Intrusion Detection Log from 07-Nov-08 to 16-Nov-11 (Entire Exercise): http://www.usma.edu/crc/sitepages/datasets.aspx; (Accessed: 17 January 2018)
24. Snort. Available at: https://www.snort.org/ (Accessed: 10 September 2017).
25. T. Jinyu and Z. Xin, "Project financing risk assessment based on ACO and SVM," 2009 ISECS International Colloquium on Computing, Communication, Control, and Management, Sanya, 2009, pp. 300-302.
26. T. Zhang and Q. Zhu, "Distributed Privacy-Preserving Collaborative Intrusion Detection Systems for VANETs," in IEEE Transactions on Signal and Information Processing over Networks, vol. 4, no. 1, pp. 148-161, March 2018.
27. W. Hu, W. Hu and S. Maybank, "AdaBoost-Based Algorithm for Network Intrusion Detection," in IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 38, no. 2, pp. 577-583, April 2008.
28. Weka, Weka 3: Data Mining Software in Java, 2017. [Online] Available at: https://www.cs.waikato.ac.nz/ml/weka/ (Accessed: October 2017).
29. X. Wang, C. Zhang and K. Zheng, "Intrusion detection algorithm based on density, cluster centers, and nearest neighbors," in China Communications, vol. 13, no. 7, pp. 24-31, July 2016.
30. Y. Freund, R. Schapire, A decision theoretic generalization of online learning and an application to boosting, J. Comput. System Sci. 55 (1) (1997) 119–139.