ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

SCIENTIFIC Research and Community

Research Article

Open Access

Toward an Appropriate Approach for Intelligent Intrusion and Detection Systems

Ouafae Elaeraj[1] and Cherkaoui Leghris[2*]

[1]LMIA, RTM Team, Faculty of Sciences and Techniques Mohammedia, Hassan II University of Casablanca, Morocco

[2]LMIA, RTM Team, Faculty of Sciences and Techniques Mohammedia, Hassan II University of Casablanca, Morocco

ABSTRACT

Now a days, the company's information security become among a main priority. Indeed, the more the attack force on the network develops, the more it is necessary to develop the security and the network surveillance. The data is to be exchanged between the internal company network and the outside one such as Internet. It is therefore necessary to be protected against malicious intrusions into the company's network, but also to monitor the traffic inside the network in order to prevent possible internal attacks.

Currently, security and reliability have become the major concerns of an individual or organization. A rule-based intrusion detection system (IDS) called Snort is an open-source software used as a network protection tool that can only detect recognized attacks. In order to detect advanced network attacks and detect fraudulent network traffic, this research paper proposes an advanced and more intelligent approach by applying machine learning. To find the best algorithm to use with Snort to improve its detection, the support vector machine (SVM) was chosen based on its accuracy. The proposed system has produced efficient detection rates versus other proposed approaches in the security intrusions detection field.

***Corresponding author**

Cherkaoui Leghris, LMIA, RTM Team, Faculty of Sciences and Techniques Mohammedia, Hassan II University of Casablanca, Morocco.
E-mail: cherkaoui.leghris@fstm.ac.ma

Received: October 31, 2022; **Accepted:** November 08, 2022; **Published:** November 22, 2022

Keywords: Security Intrusion Detection, SNORT, Machine Learning, Support Vector Machine

Introduction

The openness of systems and their interconnection with the Internet have made attacks more and more numerous and diverse. In addition to the implementation of firewalls and authentication systems, it is nowadays necessary to implement an intrusion detection system. There are two main types of intrusion detection systems. The first type is host-based intrusion detectors (HIDS), which could analyze and monitor only the activity and information of the host on which the HIDS is installed and thus ensure only the security of the host in question. The second type consists of network intrusion detectors (NIDS), which observe and analyze network traffic, look for indicators of attacks and send alerts. The most known NIDS is SNORT, which is, an open-source software, often used as a probe. It also has an active mode that allows it, when installed on a routing device, to block any suspicious traffic. It is therefore a network intrusion detector allowing real-time analysis of traffic on a network segment.

SNORT

SNORT is a free intrusion detection system (or NIDS) released under the GNU GPL. Originally written by Marty Roesch, it is currently owned by the American company Sourcefire. It is one of the most active open-source NIDS and has a large community contributing to its success. Firstly, it is open- source software that can be deployed by individuals and organizations, and it is widely used in companies. Since we are going to focus on the network security of a small or medium-sized company, we will implement a system that allows monitoring traffic and updating security rules and policies from a security manager because a small or medium-sized company may not have the means to equip itself with a "computerized information system security" department.

SNORT is a powerful Intrusion Detection System (IDS) and also an Intrusion Prevention System (IPS) that provides real-time network traffic analysis and packet logging, uses a rule-based language that combines anomaly inspection methods, protocols and signatures to detect potentially malicious activity. Using this detection system, network administrators can detect denial-of-service (DoS) attacks, distributed DoS (DDoS) attacks, common gateway interface (CGI) attacks, buffer overflows and stealth port scans. SNORT creates a series of rules that define malicious network activity, identify malicious packets, and sends alerts to users. The SNORT rule language determines what network traffic should be collected and what should happen when it detects malicious packets. This sniffing sense can be used in the same way as network sniffers and intrusion detection systems to discover malicious packets or as a complete network IPS solution that

monitors network activity and detects and blocks potential attack vectors.

How SNORT Works

Snort captures packets at a point on an IP network, analyzes the resulting flow in real time, and compares the network traffic to a database of known attacks. Known attacks are listed in rule libraries maintained by several very active communities.
Snort can also be used with other compatible modules (such as GUIs, independent attack library updaters, etc.).

Snort is compatible with most OS. Windows, Mac, Linux Ubuntu, CentOS.

SNORT Modes

SNORT allows the analysis of IP type network traffic; it can be configured to operate in three modes:

Sniffer Mode: In this mode, SNORT reads packets traveling over the network and displays them continuously on the screen; running Snort in this mode allows you to dump the data in the header and body of each packet for the screen. To start Snort so that it displays all the data in the application, enter the following: **snort -vde**

- **-v** = verbose (prints all packets)
- **-d** = display application layer data
- **-e** = display link layer packet headers

Mode "packet logger": In this mode, SNORT records network traffic in directories on disk, also allows thanks to its arguments interesting operations allowing to limit the logs to some criteria, like a range of IP address: **snort -vde -l <path_to_journal_directory>, -l** = location of log directory.

Network Intrusion Detector Mode (NIDS): In this mode, SNORT analyzes the network traffic, compares this traffic with the rules already defined by the user and the established actions to be performed; this mode allows to make a system based on IDS that will place sensors, so network cards that we will place in peripheral networks as seen in Figure 1, so that they can analyze the incoming and outgoing flows of the network. It captures all the network traffic in real time:

- **snort.conf** used in this mode to determine rule files.
- **<rules_name>**.rules used to group related rules.

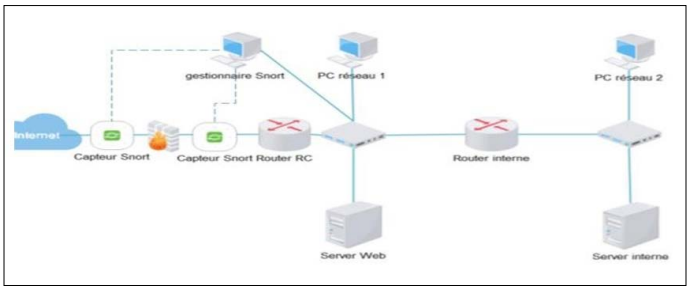

Figure 1: Operation of a NIDS

SNORT Architecture

SNORT is an open-source Network Intrusion Detection System capable of real-time analysis of traffic on IP networks. SNORT uses protocol analysis and string searching in packets for attack detection. It is used to detect a variety of attacks such as port scans, common gateway interface (CGI) attacks, buffer overflows, and more.

SNORT's architecture is organized in modules, consisting of four main modules: the packet decoder, the preprocessors, the detection engine, and the alerting and logging system as seen in Figure 2.

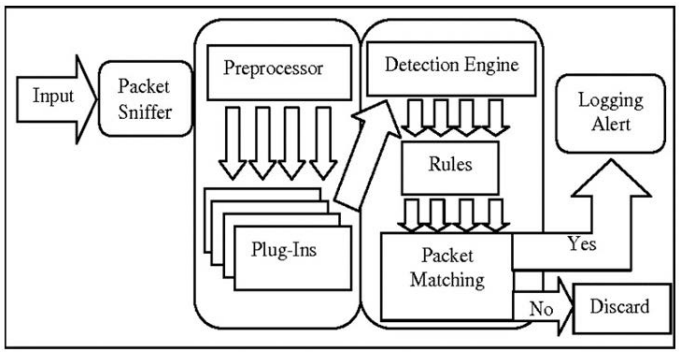

Figure 2: SNORT Architecture

The Packet Decoder: An intrusion detection system activates one or more network interfaces of the machine in promiscuous mode, this will allow it to read and analyze all the packets that pass through the communication link. SNORT uses the libpcap library to capture the frames. A packet decoder is composed of several sub-decoders which are organized by protocol (Ethernet, IP, TCP.), these decoders transform the elements of the protocols into an internal data structure;

The Preprocessors: Deals with the detection of intrusion by looking for anomalies, a preprocessor sends an alert if the packets do not respect the standards of the protocols used. A preprocessor is different from a detection rule, it is a program that aims to go into more details in the analysis of traffic. We will see through an example how preprocessors work;

Detection Engine: This is the most important part of IDS. The detection engine uses rules to detect intrusion activity. If a packet matches a rule an alert is generated. The rules are grouped into several categories in the form of files. SNORT comes with a set of predefined rules. These rules are not activated automatically, they must be activated in the snort.conf configuration file. Each rule file describes a type of traffic to be reported;

Alert and Logging System: The alert and logging system takes care of generating logs and alerts. The alerts are stored by default in the directory /var/log/Snort/. As soon as the system becomes operational, we can consult the generated alerts directly in the text files or use a management console. ACID (Analysis Console for Intrusion Detection) is an application that provides a management console and allows the visualization of alerts in graphic mode. The alerts in this case are stored in a MySQL database.

Snort Rules, Security/Monitoring Policies

Snort rules are written to ensure that security policies with respect to enterprise monitoring are respected. However, it is important to know how to write a Snort rule, because there are variables, options, and signatures to write, and we are always looking for an optimal solution.
Here is the composition of a Snort rule:

- Rule action;
- Protocol;
- Source IP Address;
- Source Port;
- Direction Operator Destination;
- Destination IP Address;

- Destination Port;
- Rule Option.

We are going to see in detail each of these points to have the best use and knowledge of a Snort rule.

The Actions of the Rules

Snort rules describe a basic action. There are several possible actions to perform on the traffic as illustrated on table 1.

Table 1: Rule Action

Action	Description
Alert	The action will generate an alert and record the packet logs
Log	The action will simply log the package
Pass	The action will ignore the package
Activate	The action will alert and then activate a dynamic rule
Dynamic	The action will remain inactive until it is activated with an activate rule, then it acts as a log action rule
Drop	The action will block and log the package
Reject	The action will block and log the package. If it is a TCP packet, the action will send a tcp reset If it is an ICMP packet, the action will display "destination port unreachable".
Sdrop	The action will block the packet but will not record the logs as for the Drop action

It can be seen in table 1 that with the action, we can indicate what Snort will do, in relation to a given event. Either it will alert in the log/alert file, or log the packets with the event information in the **log/log<number>** file, or discard it, etc.

It is important to note that only the Alert, Log, Pass, Activate and Dynamic rules are going to be useful to us because these are the rules that are going to allow us to operate as an IDS (and not as an IPS because we are only going to use the IDS part of Snort).

Following the rule, we will now describe the protocols that we can configure.

Rule Protocols

The rule must be given a protocol to monitor. Indeed, it would be very difficult and very heavy in terms of bandwidth, to observe the traffic of all the protocols, therefore, we will specify one protocol per rule. Here is the choice of protocols available:

The **UDP** protocol, also called "Best Effort", is a communication and data routing protocol, but without error control.
The **TCP** protocol is based on IP addressing and is useful for communication between two devices that both have an IP address.
The **ICMP** protocol is a protocol that will allow us to send messages on the machines of a network (for example the PING).
The **IP** protocol allows the transport of data through two machines with an IP address, and it includes ICMP, TCP and UPD.

Support Vector Machines

SVMs are a class of learning algorithms initially defined for discrimination, i.e., the prediction of a binary qualitative variable. They were then generalized to the prediction of a quantitative variable. In the case of discrimination of a dichotomous variable, they are based on the search for the optimal margin hyperplane which, when possible, correctly classifies or separates the data while being as far as possible from all the observations. The principle is thus to find a classifier, or a discrimination function, whose generalization capacity (prediction quality) is the highest possible. This approach follows directly from Vapnik's work in learning theory since 1995. It focused on the generalization (or prediction) properties of a model by controlling its complexity.

As presented in the introduction, SVM is a supervised machine learning model that is mainly used for classifications (but it can also be used for regression). The intuition behind Support Vector Machines is to simply separate data by delimiting them (creating boundaries) in order to create groups. In other words, SVMs aim at solving classification problems by finding good decision boundaries (see figure 3 below) between two sets of points belonging to two different categories.

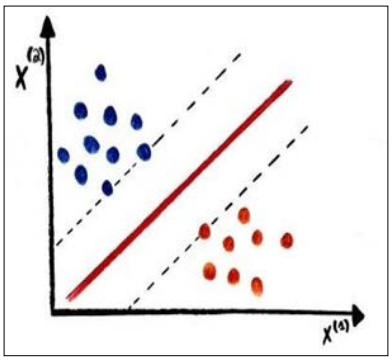

Figure 3: SVM Principle

A decision boundary can be thought of as a line or surface separating your training data into two spaces corresponding to two categories. To classify new data points, you simply check which side of the decision boundary they are on.
SVMs perform the search for these boundaries in two steps:

1. The data are mapped to a new high-dimensional representation where the decision boundary can be expressed as a hyperplane;
2. A good decision boundary a separation hyperplane is computed by trying to maximize the distance between the hyperplane and the nearest data points in each class, a step called margin maximization. This allows the boundary to fit well to new samples outside the training dataset, as in Figure 4.

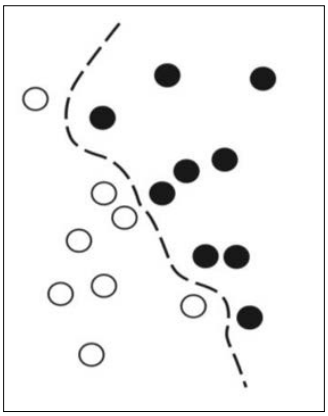

Figure 4: Decision Frontier

This technique used by Support Vector Machines is called "kernel trick". It transforms the data, and then, based on these transformations, it finds an optimal boundary between the possible

outcomes. In other words, it performs extremely complex data transformations and then determines how to separate the data based on the labels you set.

This tool is becoming widely used in many types of applications and is proving to be a serious competitor to the most powerful algorithms (model aggregation). The introduction of kernels, specifically adapted to a given problem, gives it great flexibility to adapt to very diverse situations (pattern recognition, genomic sequences, characters, spam detection, diagnostics...). Note that, algorithmically, these algorithms are more penalized by the number of observations, i.e., the number of potential support vectors, than by the number of variables. Nevertheless, powerful versions of the algorithms allow taking into account large databases in acceptable computation times.

This paper is organized as follows. Section 2 presents related work to study what others have done in the studied area, then section 3 models the proposed intelligent system that combines the Snort intrusion detection system and the SVM machine learning algorithm, as well as the results. Finally, section 4 dresses the conclusion and some prospects.

Related Works

Shah, Syed & Issac, Biju & Jacob, Seibu [1] conducted a study on machine learning algorithms that can be used with Snort by first performing an analysis via Weka on three different IDS datasets. The SVM algorithm was found to perform the best, followed by fuzzy logic and decision tree. The simple and ensemble versions of SVM were then applied to Snort to improve detection, followed by the combined versions of SVM and fuzzy logic, and SVM and decision tree. The false positive rate of the machine learning optimized intelligent intrusion detection system 17 and the false negative rate were significantly reduced in the optimized versions, especially with the firefly optimization with RPF of 9.0% and FNR of 1.7%.

The paper [2] proposes a system that uses learning rule-based classification and machine learning to automatically detect attacks against networks and computer systems more accurately, the approach uses two different learning styles in series to detect network intrusions. First, they used a rule-based system to identify incoming network packets as either intrusion or normal packets, and then a trained machine learning classifier model to further validate whether the incoming packets are intrusion or normal packets. For the "SNORT" rule-based system and machine learning classification, they used simple logistics, J48, and sequential minimum optimization (SMO). The experimental results show that the proposed approach can successfully reduce the false positive and false negative rate of rule- based NIDS.

Snort captures and checks in real time whether data packets match the traffic characteristics described by a certain detection rule and triggers an alarm if they match, due to insufficient packet capture capacity and performance flaws in Snort's detection engine module. It is difficult to process all the data packets arriving in real time when Snort uses a large number of detection rules to process high- speed network traffic. This results in a high false negative rate. Longwen Shuai & Suo Li [3] analyzed Snort's architecture and proposed a key to reducing the false negative rate under high-speed network traffic: it is to improve Snort's packet capture capability and the performance of the detection engine module. To improve the performance of Snort's packet capture module, they implement Snort's DAQ module based on the DPDK high-performance packet processing framework. Experiments show that Snort's packet capture capability and malicious traffic detection rate under high-speed network traffic were significantly improved after the optimization.

This paper [4] examines the performance of two open-source intrusion detection systems, namely Snort and Suricata, in accurately detecting malicious traffic on computer networks. These systems were installed on two different, but identical computers and performance was evaluated at a network speed of 10 Gbps. Suricata was able to handle faster network traffic than Snort with a lower packet loss rate, but it consumes more computer resources. Snort had better detection accuracy and was therefore chosen for the following experiments. It was observed that Snort triggered a high rate of false positive alarms. To solve this problem, an adaptive Snort plug-in was developed. In order to select the best performing algorithm for the adaptive plug-in of Snort, an empirical study was performed with different learning algorithms and the support vector machine (SVM) was selected. The best result was obtained using an optimized SVM with a Firefly algorithm with a FPR (false positive rate) of 8.6% and FNR (false negative rate) of 2.2%, which proves a good result.

The paper [5], proposes an SDN software-defined network based intrusion detection system using support vector machines (SVMs) along with selective logging for IP tracing which results in a detection accuracy of 95.98% obtained on the full NSL-KDD dataset and 87.74% on the selected sub- features of the dataset. The detection of abnormal traffic and network intrusion is performed at the PACKET_IN event at the controller level and then they retrieve flow statistics from OpenFlow switches at regular intervals. Selective logging of suspicious packets/flows during a PACKET_IN event which allows for IP tracing in the event of an attack that can be initiated by a network administrator using an HTTP-based web console.

An analytical study of intrusion detection techniques presented on the paper [6] the latter is based on support vector machine (SVM) consisting of four main steps, namely data collection, preprocessing, SVM technique for training and testing and decision. The simulated results were analyzed on the basis of overall detection accuracy, receiver operating characteristic, and confusion matrix (ROC). The NSL-KDD dataset is used to analyze the performance of SVM techniques. The results obtained show that the linear SVM, quadratic SVM, fine Gaussian SVM and mean Gaussian SVM give an overall detection accuracy of 96.1%, 98.6%, 98.7% and 98.5% respectively.

Distributed Denial of Service (DDoS) attacks are common today, especially due to the relative simplicity of their implementation and their effectiveness against an unprepared target. These attacks can cause significant financial losses through service interruption or indirectly through damage to the target's image. The occurrence of Software Defined Network (SDN) evokes new methods such as deep learning algorithm is adopted to model the attack behavior based on the collection from the SDN controller. In this paper [7], a mininet and projector simulation platform of SDN environment is built, the characteristic values at 6 tuples of the switch flow table are extracted, and then the DDoS attack model is built by combining the SVM classification algorithms. Experiments show that the average accuracy rate of 95.24%.

The support vector machine (SVM) has played an important role in providing potential solutions to the IDS problem. However, the feasibility of introducing SVM is affected by the difficulties in selecting the appropriate kernel and its parameters. The paper

[8] presents a work to apply different kernels for SVM in ID Son the KDD'99 and NSL-KDD dataset as well as to determine which kernel is the best for SVM. An RRE-KDD dataset used to remove redundant records from KDD'99 train. This RRE- KDD includes both the KDD99Train+ and KDD99 Test+ datasets for training and testing purposes, respectively. The results show that the kernel can achieve higher detection rate and lower false positive rate with higher accuracy than the other kernels in the RRE-KDD and NSL-KDD datasets.This paper [9] proposes an anomaly-based IDS using genetic algorithm and support vector machine (SVM) with a new feature selection method. This model uses a feature selection method based on the genetic algorithm by reducing the data dimension, increases the detection of true positives and simultaneously decreases the detection of false positives. The results show that the proposed method can simultaneously achieve high accuracy and low false positive rate (FPR) which results in more stable features compared to other techniques. The proposed model is experimented and tested on the KDD CUP 99 and UNSW-NB15 datasets.

Proposed System

In order to improve the intrusion detection capabilities, we propose in this paper, a system based on the fusion of Snort and SVM techniques. We conducted several platform-based experiments with the hardware as described below.

Architecture

Our platform requires a physical machine to generate the legitimate traffic with two virtual machines as shown in Figure 5. The first virtual contains the proposed detection system with SVM algorithm in fusion with Snort IDS which uses the default rule set. Using another physical machine, the malicious traffic is generated. Some configured software details are shown in Table 2.

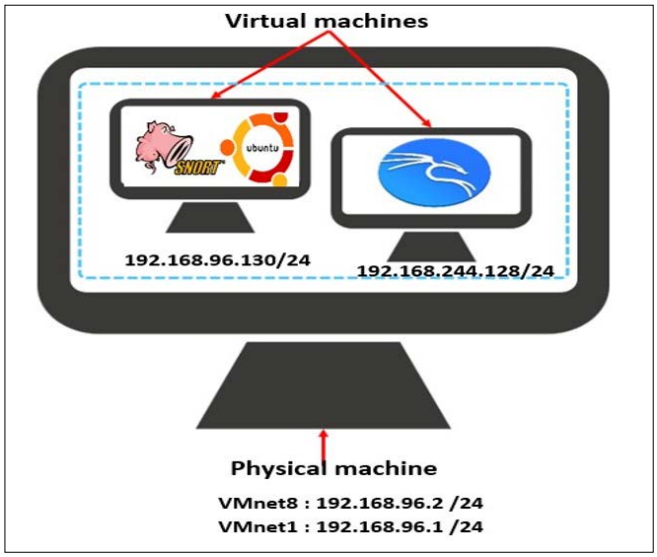

Figure 5: Platform Architecture

Table 2: Experiment Network Specifications

Machine/Software Type	Specification	Tools Used
Ubuntu 20.04.2	Virtual Machine, 4 cores CPU,8 GB Memory, 20 GB Hard Disk	Snort 2.9.17 IDS; Snort logs, Jupyter notebook
Kali Linux	Virtual Machine, 1 cores CPU,1 GB Memory, 10 GB Hard Disk	Malicious Network Traffic Generator
User PC /**Windows 8**	Physical machine	Legitimate Network Traffic Generator

The proposed intrusion detection system has been divided into four main phases, each of which is illustrated in Figure 6.

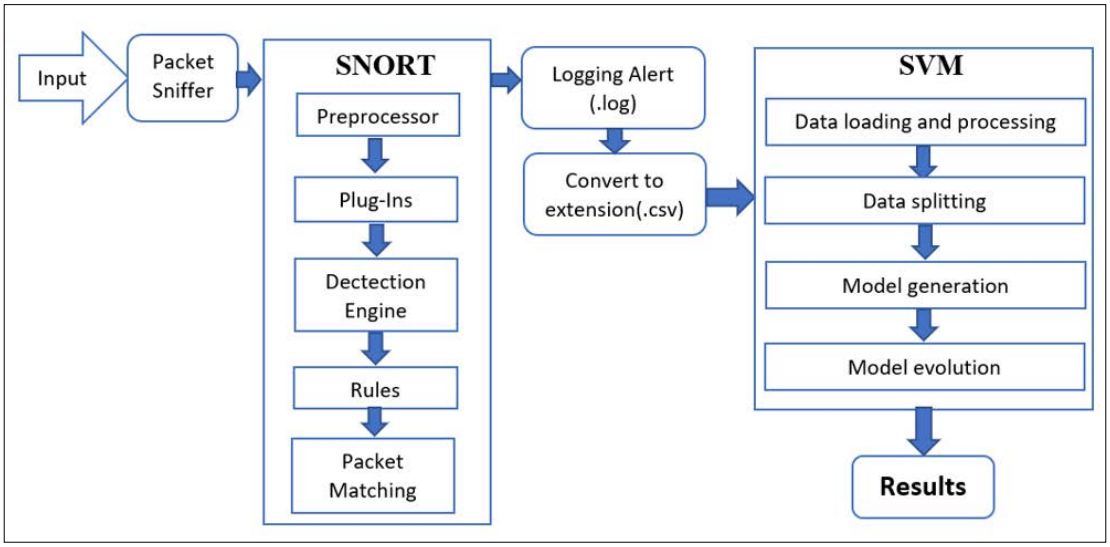

Figure 6: Proposed architecture of an IDS

The role of each phase is described as follows

Input

This is the first phase where legitimate and malicious network traffic occurs via two generators, the first one via Windows physical machine and the second one via Kali Linux Metasploit framework.

The latter generated 3 types of malicious traffic like SSH, HTTP and ICMP. The Metasploit framework generates malicious traffic with different exploits and payloads for different operating systems.

SNORT

Snort IDS inspect traffic and trigger alarms when incoming traffic matches the rule set. Common performance metrics for IDS detection accuracy are :

- 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): The false is identified as true.

The basic steps for running a SNORT intrusion detection system are as follows:

SNORT Configuration

Snort has several configuration files. The most important ones are: /etc/snort/snort.conf (the main configuration file) and the .rules files located in the /etc/snort/rules/ directory.

The snort.conf configuration file contains six basic sections:

- Variable Definitions: defines various variables that are used in Snort rules as well as for other purposes, such as specifying the location of rule files ;
- Configuring Dynamic Libraries ;
- Preprocessor configuration: Preprocessors are used to perform actions before a packet is evaluated by the main Snort detection engine ;
- Output module configuration: Output modules control how Snort data will be recorded ;
- Defining new action types: You can define custom action types in the Snort configuration file;
- Rules configuration and include files: possibility to add any rules in the main snort.conf file, the convention is to use separate files for the rules. These files are then included in the main configuration file using the include keyword. These rules determine how Snort reacts when it is started. There is a file called local.rules which allows the network administrator to define his own rules.

The SNORT Rules

SNORT rules are composed of two distinct parts:

- **The header** allows to specify the type of alert to generate (alert, log, and pass) and to indicate the basic fields necessary for the filtering: the protocol as well as the source, destination IP addresses and ports ;
- **The options** allow us to refine the analysis by breaking down the signature into different values to be observed among certain header fields or among the data.

Snort will inspect legitimate and malicious traffic and trigger alarms when the incoming traffic matches the rule set. The number of common rule sets used in our experiment is shown in Figure 7 and are described as follow.

```
# LOCAL RULES
#-------------

alert icmp any any -> any any (msg:"test ICMP";sid:10000001;)
alert tcp any any -> any any (msg:"Alert FTP "; sid:100006927; rev:003;)
```

Figure 7: Local Rules

- **alert icmp any any - > any any (msg: "test ICMP "; sid: 10000001;)**

This rule alerts when there is an ICMP packet (ping traffic). The sid (signature id) keyword is used to uniquely identify Snort rules. The turn keyword is used to uniquely identify revisions of Snort rules.

- **alert tcp any any - > any any (msg: "Alert FTP"; sid: 100006927; rev: 003;)**

This rule alerts when there is TCP traffic, the turn keyword is used to uniquely identify revisions of Snort rules.

SNORT Configuration File

For the snort configuration, we will edit the snort.conf file (/etc/snort/snort.conf).

We have only modified the first part of the file which concerns the network variables. Here are the modifications we have made in this file such as illustrated on the figure 8:

✓ **var HOME_NET any/* indicates the address of the network interface that listens for traffic.**

The default value is any. We can customize it by using the IP address of the interface or network to protect.

✓ **var EXTERNAL_NET any/* indicates the external network(s) to listen to.**

The default value is any, which means that traffic from any network is analysed. To exclude the network to be protected we use !HOME_NET instead of any. You can also specify the networks by using: [network address1, network address2 ...].

✓ **var RULE_PATH /etc/snort/rules** / here we specify the directory where the .rules files are located.

```
##################################################
# Step #1: Set the network variables.  For more information, see README.variables
##################################################

# Setup the network addresses you are protecting
ipvar 192.168.96.130/16

# Set up the external network addresses. Leave as "any" in most situations
ipvar EXTERNAL_NET any

# List of DNS servers on your network
ipvar DNS_SERVERS any
```

Figure 8: Snort Configuration File

SNORT Filtering

We start snort filtering with the bellow command. This is illustrated on the Figure 9.

sudo snort -A console -i ens38 -u snort -c /etc/snort/snort.conf

```
root@ubuntu:/var/log/snort# sudo snort -A console -i ens38 -u snort -g snort -c /etc/snort/snort.conf
Running in IDS mode

        --== Initializing Snort ==--
Initializing Output Plugins!
Initializing Preprocessors!
```

Figure 9: SNORT Filter Command

Conversion

After, we convert the encrypted snort log file to a file with .csv extension to be able to integrate it to the SVM algorithm as data:

Tcpdump -n -tttt -r snort.log.xxxxxxxxxx > /home/ouafae/data.csv

The Proposed SVM Algorithm

Data Loading and Processing

At first, we call all the necessary library packages like Sklearn, Pandas, etc.

Then, we do the import of the dataset which are log files of SNORT converts to a .csv file which contains all the input traffic alarms like false positive, false negative and true positive.

Figure 10 shows the information of the input data of the dataset such as date, time, source address, destination address, source port, destination port, and protocol.

	DATE	TIME	IP	source.address	portsrc		address.destination	portdes	ack	win	lenght
0	2022-02-11	08:20:29	IP	192.168.96.128	51870	>	34.107.221.82	80	ack2059173696,	win64020,	length0
1	2022-02-11	08:20:29	IP	192.168.96.128	51870	>	34.107.221.82	80	ack2059173696,	win64020,	length0
2	2022-02-11	08:20:29	IP	192.168.96.128	51870	>	34.107.221.82	80	ack2059173696,	win64020,	length0
3	2022-02-11	08:20:29	IP	192.168.96.128	51870	>	34.107.221.82	80	ack2059173696,	win64020,	length0
4	2022-02-11	08:20:29	IP	192.168.96.128	51870	>	34.107.221.82	80	ack2059173696,	win64020,	length0
5	2022-02-11	08:20:30	IP	192.168.96.128	51870	>	34.107.221.82	80	ack1,	win64020,	length0
6	2022-02-11	08:20:30	IP	34.107.221.82	80	>	192.168.96.128	51870	ack1,	win64240,	length0
7	2022-02-11	08:20:30	IP	192.168.96.128	51870	>	34.107.221.82	80	ack1,	win64020,	length0
8	2022-02-11	08:20:30	IP	34.107.221.82	80	>	192.168.96.128	51870	ack1,	win64240,	length0
9	2022-02-11	08:20:30	IP	192.168.96.128	51870	>	34.107.221.82	80	ack1,	win64020,	length0

Figure 10: Dataset

The analysis will be based on 3 characteristics: source address and destination address and destination port number.

Data Splitting

To understand the performance of the model, we need to split the dataset into a training set and a test set as shown in Figure 11.

Splitting the dataset using the train_test_split () function contains three parameters features (source address, destination address, portsrc and portdest), target (wine_dataset.label) and test_set size.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(wine_dataset[['source.address', 'adress.destination',
'portsrc', 'portdes']],wine_dataset.label, test_size = 0.3)
```

Figure 11: Data Splitting

Model Generation

Let's build a support vector machine model. First, we imported the SVM module and create a support vector classifier object by passing the kernel argument as a linear kernel in the SVC ().

One of the important parameters is the kernel, which is a function that is used to transform the data into a specific representation. SVMs use different types of kernel functions, for example, linear, nonlinear, polynomial, Radial Basis Function (RBF) and sigmoid. The polynomial and RBF are useful for the nonlinear hyperplane. The polynomial and RBF kernels compute the dividing line in the higher dimension. In some applications, it is suggested to use a more complex kernel to separate curved or nonlinear classes. This transformation can lead to more accurate classifiers.

RBF is the default kernel used in the sklearn SVM classification algorithm and can be described

$$K(x, x') = e^{-\gamma||x-x'||^2}$$

with the following formula:

Where gamma can be set manually and must be >0. The default value for gamma in the sklearn SVM classification algorithm is:

$$\gamma = \frac{1}{n\,features * \sigma^2}$$

$||x - x'||^2$ is the squared Euclidean distance between two feature vectors. **Gamma** is a scalar that defines the influence of a single training example.

Thus, given the above setup, we can control the influence of individual points on the overall algorithm. The larger the gamma, the closer the other points must be to affect the model.

In our case, we will choose an RBF kernel with the parameters random state =1, gamma=0.05 and C=0.1 such as in the Figure 12. The parameter C, common to all SVM kernels, compensates for the misclassification of the training examples by the simplicity of the decision surface. A low C keeps the decision surface smooth, while a high C ensures that all training examples are correctly classified. In addition, we can use random_state to select records at random.

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='rbf', random_state=1, gamma=0.05, C=0.1)
```

Figure 12: Model Generation

Model Evolution
The fit method of the SVC class is called to train the algorithm on the training data, which is passed as a parameter to the fit method. To make predictions, the predict method of the SVC class is used Figure 13.

```
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)
```

Figure 13: Model Evolution

Results
Confusion matrix and accuracy are the most commonly used measures for classification tasks. Precision can be calculated by comparing the actual values of the test set with the predicted values.

The Scikit-Learn metrics library contains the methods classification_report and confusion_matrix as in Figure 14, which can be easily used to find out the values of these important metrics.

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Figure 14: Printing the Confusion Matrix

The accuracy it generates is 99%, i.e., the algorithm learns the patterns in the dataset with 99% accuracy with a rate of true positive 162, false positive 1, true negative 160 and null value of false negatives as can be seen in Figure 15.
The confusion matrix shows the relationship between the predicted result and the expected results.

```
[[162   0]
 [  1 160]]
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       162
           1       1.00      0.99      1.00       161

    accuracy                           1.00       323
   macro avg       1.00      1.00      1.00       323
weighted avg       1.00      1.00      1.00       323
```

Figure 15: Results

Precision: Is the ability of a classifier not to label a positive instance that is actually negative. For each class, it is defined as the ratio of true positives to the sum of true and false positives.
Precision: = TP/ (TP + FP)
Recall: Recall is the ability of a classifier to find all positive instances. For each class, it is defined as the ratio of true positives to the sum of true positives and false negatives.
Recall= TP/ (TP+FN)
F1 score: is a weighted harmonic average of precision and recall such that the best score is 1.0 and the worst is 0.0. In general, F1

scores are inferior to precision measures because they incorporate precision and recall in their calculation. As a general rule, the weighted average of F1 should be used to compare classifier models, not the overall precision.

F1 score = 2*(Recall * Precision) / (Recall + Precision)

Support: The number of occurrences of each label in y_test.

Conclusion

The security of computer networks remains a very sensitive subject for the actors of the computer world because the variables that revolve around this subject are often difficult to master. Even if the evolution of technology has improved, the security mechanisms in computer networks is still difficult or impossible to guarantee 100% security.

Intrusion detection systems are very effective in monitoring and detecting data packets in network traffic. This work focuses on combining SVM machine learning algorithm with Snort intrusion detection system against different types of threats. This proposed intrusion detection system shows better results by comparing it with several support vector machine (SVM) based intrusion detection works, like the work presented on the paper [6] which analyzes the performance of SVM techniques with NSL-KDD dataset and gives maximum detection accuracy 98.5%. Finally we can conclude from our proposed system that Support Vector Machine gives better accuracy on the imported dataset of SNORT.

In future work, advanced data mining techniques and machine learning techniques will be used to detect new suspicious attacks on huge amount of data.

References

1. Shah Syed, Issac Biju, Jacob Seibu (2018) Intelligent Intrusion Detection System Through Combined and Optimized Machine Learning. International Journal of Computational Intelligence and Applications 17.
2. Aslam Urooj, Batool Ezzat, Ahsan Syed, Sultan Abdullah (2017) Hybrid Network Intrusion Detection System Using Machine Learning Classification and Rule Based Learning System International Journal of Grid and Distributed Computing 10:51-62.
3. Shuai Longwen, Li Suo (2021) Performance optimization of Snort based on DPDK and Hyperscan Procedia Computer Science 183:837-843.
4. Shah Syed, Issac Biju (2018) Performance Comparison of Intrusion Detection Systems and Application of Machine Learning to Snort System. Future Generation Computer Systems 80:157- 170.
5. Hadem Pynbianglut, Saikia Dilip, Moulik Soumen (2021) An SDN-based Intrusion Detection System using SVM with Selective Logging for IP Traceback Computer Networks 191.
6. Bhati Bhoopesh, Rai C (2019) Analysis of Support Vector Machine-based Intrusion Detection Techniques. Arabian Journal for Science and Engineering 45.
7. Ye Jin, Cheng Xiangyang, Zhu Jian, Feng Luting & Song Ling (2018) A DDoS Attack Detection Method Based on SVM in Software Defined Network. Security and Communication Networks 2018:1-8.
8. Hasan Md. Al, Xu Shuxiang, Kabir Mir, Ahmad Shamim (2016) Performance Evaluation of Different Kernels for Support Vector Machine Used in Intrusion Detection System International journal of Computer Networks & Communications 8:39-53.
9. Gharaee Hossein, Hosseinvand Hamid (2016) A new feature selection IDS based on genetic algorithm and SVM 139-144.