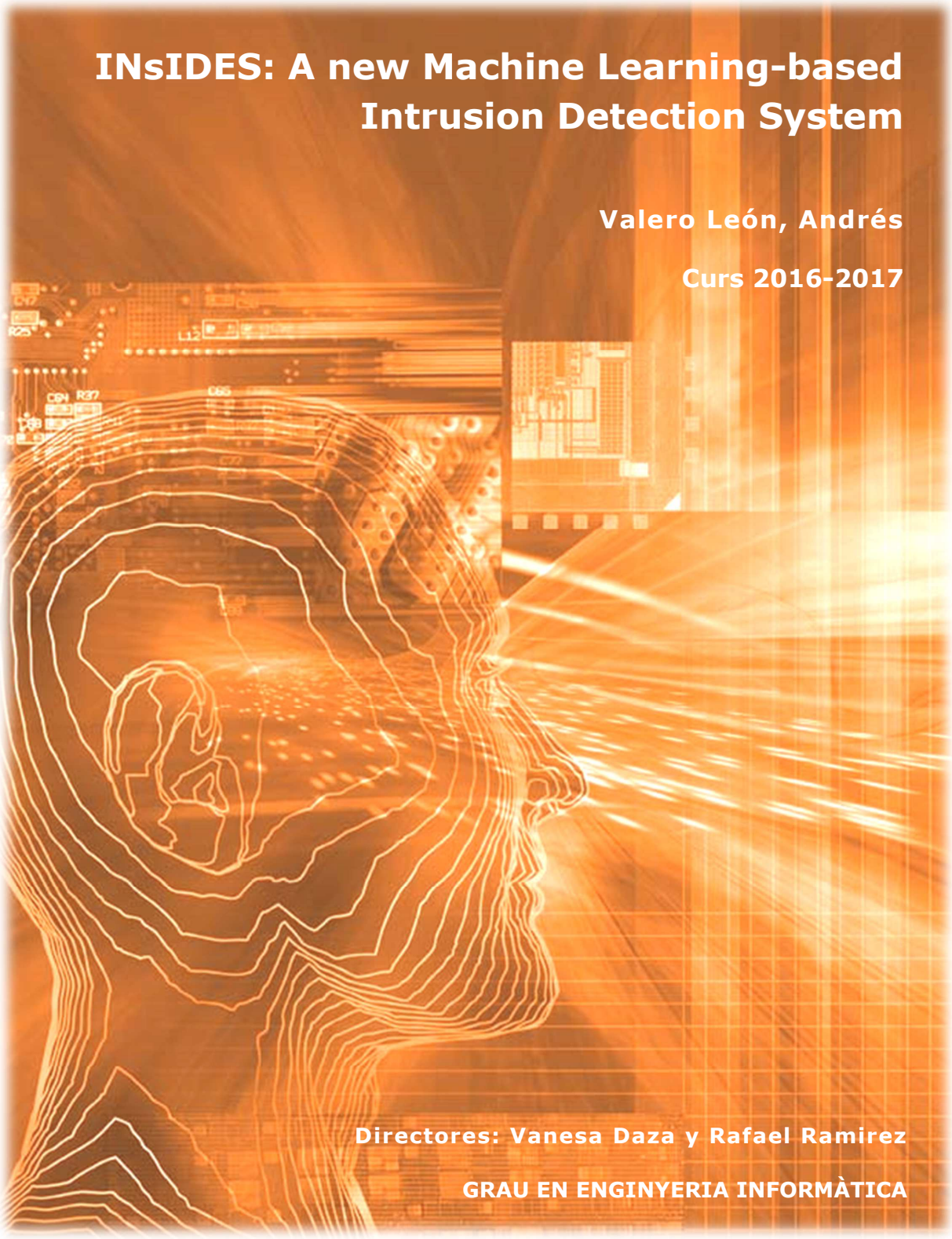
INsIDES: A new Machine Learning-based Intrusion Detection System

Valero León, Andrés

Curs 2016-2017



Directores: Vanesa Daza y Rafael Ramirez

GRAU EN ENGINYERIA INFORMÀTICA

upf. Universitat Pompeu Fabra Barcelona
Escola Superior Politècnica

Treball de Fi de Grau

Acknowledgments

I would like to thank Vanesa Daza and Rafael Ramirez for they helpful contributions and they support to this project. Also, I extend my gratitude to Nour Moustafa and Jill Slay, authors of the UNSW-NB15 dataset, for give me the instructions to calculate the ct_state_ttl feature.

Abstract

Nowadays there are an enormous number of attacks over the Internet that makes our information to be continuously at risk. Intrusion Detection Systems (IDS) are used as a second line of defense. They observe suspicious actions in the network to detect attacks. One of the most popular ones is Snort. It is an open source IDS and the rules to detect the attacks are updated offline. As there are new types of attacks almost every day, it has a low detection rate especially for new types of attack. The aim of this work is to create an IDS using machine learning techniques in order to be more efficient detecting attacks than Snort. The proposed IDS is compared with Snort using the newly UNSW-NB15 dataset. The results show a detection rate of 98.11% and a false alarm rate of 8.57% for INsIDES, whereas Snort has a detection rate of 2.43% and a false alarm rate of 30.66%, assuring encouraging trails when machine learning techniques are applied to traditional rule-based IDS.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The number of attacks over the Internet are incrementing each day. In recent weeks, computer hackers have attacked several companies around the world with a ransomware called WannaCry. This incident evidences that today more than ever the information of individuals and organizations are continuously at risk. Numerous traditional defenses employed by organizations like antivirus, anti-spam and firewalls in order to prevent the attacks are unable to detect such sophisticated ones. For this reason, Intrusion Detection System (IDS) began to appear as a second line of security monitoring the network activity trying to detect an intrusion.

The main problems with current intrusion detection system are efficiency and accuracy of detecting intrusions. Snort nowadays is commonly used misuse detection system model. It is effective against known attacks but is mostly ineffective against novel attacks. In addition , the current intrusion detection systems are less effective with real time detection of unknown attacks. On the other hand anomaly detection system is distinct from misuse detectors, it can detect unknown attacks. However, anomaly detectors suffer from a high false alarm rate [1]

This project proposed a new machine learning-based IDS software using the anomaly detection and flow-based approach. We extract the best features using a partition of the UNSW-NB15 dataset with the Weka software. Then we train and test several classificators in order to get the best

one to classify the network events. Subsequently, we integrate this classificator into a Java program, that we called INsIDES, and we tested it with the entire dataset. Finally, we compare our results with Snort showing that our software gets a detection rate of 98.11% and false alarm rate of 8.57% whereas Snort gets 2.43% detection rate and 30.66% of false alarm rate.

The rest of the project is organized as follows: in chapter 2 we introduce some backgrounds concepts of cyber security and machine learning and also review some related works to this project. In chapter 3 we show an explain the different materials used in this project. Then in chapter 4, we explain the methodology that we applied to make INsIDES and the comparison with Snort. After that, in the chapter 5, we show and discuss the results of the different methodologies applied and the results of the comparison between InSIDES and Snort. Finally, in chapter 6, we conclude the project and point some future work that we are planning to make.

Chapter 2

Background

2.1 Cyber Security

Cyber security[2] is a field of computing that aims to protect the computers systems from the theft or damage to their hardware, software or information, as well as from disruption or misdirection of the services they provide. It includes controlling physical access to the hardware, as well as protecting against harm that may come via network access, data and code injection. This field is growing importance due to the increasing reliance on computer systems and the Internet. According to the last report of Symantec[3], also helps the fact that attacks over the Internet are growing every year increasing the data breaches of individuals and organizations as we can see in figure 2.1.

In the same report they indicate that cyber attackers revealed new levels of ambition in 2016, a year marked by extraordinary attacks, including multi-million dollar virtual bank heists, overt attempts to disrupt the US electoral process by state-sponsored groups, and some of the biggest distributed denial of service (DDoS) attacks on record powered by a botnet of Internet of Things devices.

Seeing the increase of attacks our project intends to add a powerful tool to try to stop attacks as much as possible.

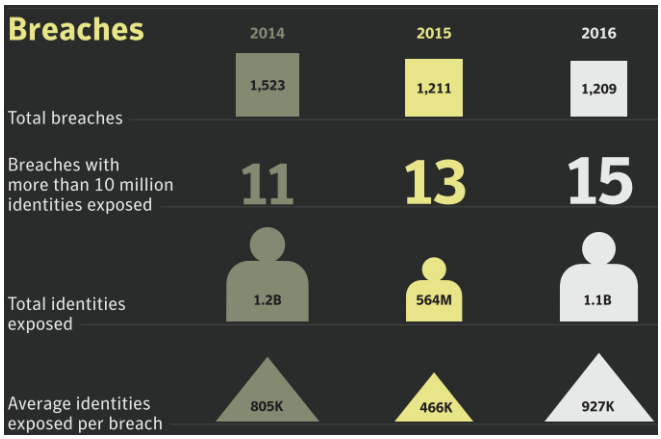


Figure 2.1: Data breaches in the past 3 years

2.1.1 Vulnerabilities and Attacks

A vulnerability is a flaw in the system through which an attacker can take advantage of it and compromise the system. In order to secure a computer system we need to know and understand the attacks that can be made against it. These attacks can typically be classified into one of the attacks families below[2][4]:

- **Physical attacks:** attacks focused in damage the computer or network hardware. These are the most difficult due to the fact that you need physical access to the system but it is the most effective.
- **Buffer Overflows:** attacks that gain control or crash a process on the target system by overflowing a buffer of that process.
- **Password attacks:** attacks trying to gain credentials access for a protected system. One example it could be the bruteforce attack to guess the password of a user.
- **Backdoor:** is any secret method of bypassing normal authentication or encryption in a computer system.

- **Exploits:** is a piece of software that takes advantage of a vulnerability in order to cause unintended behavior to occur on computer software.
- **Denial of Service:** as the name suggests, this attack denies the service to legitimate users making more requests to the service than it can handle. Usually the attack come from a large number of points which implies that defending will be more difficult.
- **Worms:**a program that self-propagates across a network. Self-propagation is the characteristic that differentiates worms from viruses. Inside this family are the malwares, ransomwares, and so on.
- **Viruses:** a virus is regarded as a worm that only replicates on the infected host computer. Hence, it needs user interactions to propagate to other hosts.
- **Information Gathering:** an attack that does not directly damage the target system, but gains information about the system, possibly to be used for further attacks in the future like port or service scanning.
- **Social Engineering:** this kind of attack attempts to convince the user to do an action that will allow the attacker to gather personal information such as passwords or compromise the system. This attack is usually the most effective due to the fact that the user is the weakest layer in a computer system.

2.1.2 Security Defenses

Typically in order to prevent or combat the attacks organizations apply what is called defense in depth[5]. This concept consist in place multiple layers of security defenses throughout a computer system. Usually these layers are[2][5]:

- **Computer access control:** this layer includes everything related to identification, authorization, authentication, access approval and audit of the system.
- **Application Security**: this layer includes all the applications that combat attacks like antivirus, antimalware, and so on. It also includes how a operating system it has to be secured and also good practices of secure coding and design.
- **Firewall:** is a software or hardware system that monitors and controls the incoming and outgoing network traffic based on predetermined security rules. The last generation firewalls can control up to the application layer.
- **IDS:** is hardware, software, policy or their combination responsible for uncovering the possible intrusions from the network audit data.
- **IPS:** Some IDS have the ability to respond to detected intrusions. Systems with response capabilities are typically referred to as an intrusion prevention system (IPS).

The more layers we put in the system, the more secure the system will be. However, no system can be made perfectly secure because of financial or complexity constraints, hence the attacker will eventually find a way to break into our system and we need to be ready for this situation with countermeasures like a robust and proved contingency plan.

2.1.3 Intrusion Detection System

An intrusion detection system (IDS) is a device or software application that monitors a network or systems for malicious activity or policy violations. Any detected activity or violation is typically reported to an administrator.

The most common classifications of an IDS are network intrusion detection systems (NIDS) and host-based intrusion detection systems (HIDS).

A HIDS monitors only the files of an individual operating system for suspicious activity, whereas a NIDS analyzes the network traffic for the same purpose. Hence, NIDS can protect all the network devices in the network, whereas HIDS only can protect the machine where it its installed.

Detection Methods

Nowadays exists three methods through which an IDS can detect an intrusion: misuse detection, anomaly detection or hybrid detection:

- **Misuse[6]:** also called signature-based, it attempts to model abnormal behavior and matching the network traffic against a signature base of known attacks therefore any match of which clearly indicates system abuse. The advantage of misuse detection over anomaly detection is higher accuracy and lesser false alarms for the known attacks. The problems with misuse detection models is how to represent the signatures of all possible attacks and how to write signatures that are very different from the normal data pattern. Other problem implicit to the misuse detection model is how to update the signature base when newer attacks appear.

- **Anomaly[6]:** an anomaly detection attempts to model normal behavior of a network.This technique observes the user behavior over the period of time and builds the model that closely represents the network legitimate behavior. Events which are different from this model are considered as an attack. The problem with anomaly detection model is how to define a model for normal behavior and how to handle evolving normal network behavior. To solve this problem, our approach utilize machine learning techniques helping us to make a model for normal behavior of the network.

- **Hybrid[6]:** the fusion of misuse and anomaly detection techniques is called hybrid approach. It combines the positive aspects of both approaches. However, the problem is the added complexity to lay down the two approaches together to form a complex system.

Packet Inspection vs Flow-based

In an attempt to find known attacks or unusual behavior, IDS traditionally inspect the contents of every packet. As stated in[4], the problem of packet inspection, however, is that it is hard, or even impossible, to perform it at the speed of multiple Gigabits per second. For high-speed lines, it is therefore important to investigate alternatives to packet inspection. One option that currently attracts the attention of researchers and operators is flow-based intrusion detection. With such approach, the communication patterns within the network are analyzed, instead of the contents of individual packets.

This project uses this new approach selecting only the features related to the network flow and compares it with Snort, a packet inspection and flow-based solution.

2.2 Machine Learning

Machine learning is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. It explores the construction and study of algorithms that can learn from, and make predictions on data. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions, rather than following strictly static program instructions.[6] Intrusion detection model is a multinomial classifier problem that can classify network events as normal or attack events, such as Denial of Service (DOS) or Botnets.[6]

Machine learning techniques can be classified as: supervised and unsupervised. The supervised approach consists to make predictions by using the characteristics of the known data. Whereas unsupervised learning aims to find a way to organize the unknown data.

In this project, supervised machine learning is used, specifically, the multi-class classification because we have to know to which class a network event belongs to. There are several algorithms for classification in machine learning, but in this research four of them are used.

2.2.1 Support Vector Machines (SVM)

SVM transforms the data in higher dimensions and finds the hyperplane that best separates the data. A support vector machine is based on the notion of the margin and tries to find the maximum margin between the dataset. SVMs revolve around the notion of a margin either side of a hyperplane that separates two data classes. Maximizing the margin and thereby creating the largest possible distance between the separating hyperplane and the instances on either side of it has been proven to reduce an upper bound on the expected generalization error.[6]

2.2.2 K-Nearest Neighbors

K-nearest neighbor computes the approximate distances between different points on the input vectors, and then assigns the unlabeled point to the class of its K-nearest neighbors. In the process of create k-NN classifier, k is an important parameter and different k values will cause different performances. If k is considerably huge, the neighbors which used for prediction will make large classification time and influence the accuracy of prediction. k-NN does not contain the model training stage, but only searches the examples of input vectors and classifies new instances. Therefore, k-NN trains online the examples and finds out k-nearest neighbor of the new instance.[7]

2.2.3 Ripper

RIPPER induces rules directly from the training data by using a separate-and-conquer approach. It learns one rule at a time in such a way that the rule covers a maximal set of examples in the current training set. The rule is pruned to maximize a desired performance measure. All of the examples that are correctly labeled by the resulting rule are eliminated from the training set. The process is repeated until the training set becomes empty or a predetermined stopping criterion is reached.[8]

2.2.4 Decision Trees

Decision trees are trees that classify instances by sorting them based on feature values. Each node in a decision tree represents a feature in an instance to be classified, and each branch represents a value that the node can assume. Instances are classified starting at the root node and sorted based on their feature values. At each level of the decision tree a feature that best divides the tree into subclasses is selected by a variety of ways based on Entropy or Information gain. The division of the tree continues as long as any of the following condition is not met.[6]

2.3 Related Works

Yu-Xin Ding *et al.*, Min Xiao *et al.* and Ai-Wu Liu *et al.*[9] proposed a snort-based hybrid intrusion detection system using frequent episode rules and the 10% of the KDD99 Cup dataset. They create an anomaly detection module for Snort that can detect the unknown attacks and a signature generation module that extracts the signature of attacks that are detected by ADS module, and maps the signatures into snort rules. They achieve an average of detection rate of 94.07%.

Divyatmika *et al.* and Manasa Sreekesh *et al.*[10] proposed a two-tier network based intrusion detection. The tier one uses misuse detection using the MLP algorithm. The tier two uses anomaly detection using Reinforcement algorithm where network agents learn from the environment and take decisions accordingly. They used the NSL-KDD dataset. They achieve a TP rate of 0.99 and false positive rate of 0.01.

Naser Fallahi *et al.*, Ashkan Sami *et al.* and Morteza Tajbakhsh*et al.*[11] created and automated flow-based rule generator for network intrusion detection systems. [5] This intrusion detection is based on data flow rather than per-packet data like the other ones. This approach is implemented using Ripper and C5.0 algorithms to generate the rules and the ISCX 2012 dataset adding eight more features. Their results showed an average of precision of 97.20% using only flow-based features.

All of them are using an obsolete dataset that does not represent the

behavior of the modern attacks.[12][13] For this reason, we have selected the UNSW-NB15 dataset, a more recent dataset.

Chapter 3

Materials

3.1 UNSW-NB15 Dataset

This research use the UNSW-NB15[14] benchmark data that was created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) for generating a hybrid of real modern normal activities and synthetic contemporary attack behaviors. They also used the tool tcpdump to capture all the raw packet data reaching to capture more than 100 GB and 2,540,044 records. Each record has 49 features including the class label. These features were created with the tools Argus, Bro-IDS and twelve algorithms. This dataset has nine types of attacks, namely: Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms. In the table 3.1 it can be seen the distribution of each class in the dataset.

Type	No. Records	Description
Normal	2,218,761	Natural transaction data.
Fuzzers	24,246	Attempting to cause a program or network suspended by feeding it the randomly generated data.

Analysis	2,677	It contains different attacks of port scan, spam and html files penetrations.
Backdoors	2,329	A technique in which a system security mechanism is bypassed stealthily to access a computer or its data.
DoS	16,353	A malicious attempt to make a server or a network resource unavailable to users, usually by temporarily interrupting or suspending the services of a host connected to the Internet.
Exploits	44,525	The attacker knows of a security problem within an operating system or a piece of software and leverages that knowledge by exploiting the vulnerability.
Generic	215,481	A technique works against all block-ciphers (with a given block and key size), without consideration about the structure of the block-cipher.
Reconnaissance	13,987	Contains all strikes that can simulate attacks that gather information.
Shellcode	1,511	A small piece of code used as the payload in the exploitation of software vulnerability.

Worms	174	Attacker replicates itself in order to spread to other computers. Often, it uses a computer network to spread itself, relying on security failures on the target computer to access it.

Table 3.1: Full UNSW-NB15 dataset distribution.

A partition from this dataset[15] was configured as a training set and testing set. The training set has 175,341 records and the testing set has 82,332 records. This partition has 45 features including the class label. Table 3.2 below shows the distribution of records for each class label.

Type	**No. Records**
Normal	56,000
Fuzzers	18,184
Analysis	2,000
Backdoors	1,746
DoS	12,264
Exploits	33,393
Generic	40,000
Reconnaissance	10,491
Shellcode	1,133
Worms	130

Table 3.2: Portion of UNSW-NB15 dataset distribution.

We use this partition to create our classificator with Weka and the entire dataset to compare INsIDES with Snort.

3.2 Weka

Waikato Environment for Knowledge Analysis(Weka)[16] is an open source software that has a collection of machine learning algorithms for data

mining tasks. These algorithms can be applied directly to a dataset or called from the API. Additionally, Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization.

We use this software to apply the machine learning algorithms that will allow us select the best features of the dataset using feature selection and train and test several classificators to include the best one into our INsIDES software.

3.3 Argus

Argus[17] is an open source project, the network audit record generation and utilization system. Argus, itself, is next-generation network flow technology, processing packets, either on the wire or in captures, into advanced network flow data. Argus is composed of an advanced comprehensive network flow data generator, the Argus sensor, which processes packets and generates detailed network flow status reports of all the flows in the packet stream.

We have integrated this software in the version 3.0.8.2 into INsIDES for the purpose to be able to extract the selected features from the raw packet dataset in order to create the input instances to our classificator.

3.4 Tcpreplay

Tcpreplay[18] is a suite of free open source utilities for editing and replaying previously captured network traffic. We use this software to inject the packets through the network interface for the purpose to simulate a real time network intrusion detection system.

3.5 Snort

Snort[19] is an open source network intrusion prevention system, capable of performing real-time traffic analysis and packet logging on IP net-

works. It can perform protocol analysis, content searching/matching, and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more.

We have selected Snort to compare the results of INsIDES with one of the most used IDS in the market. We installed the Snort version 2.9.9.0 and the lastest rules available in 2017-04-30: community and registered. We use Snort software together with the software Barnyard2[20] to allow him write to the MySQL database in an efficient manner.

Chapter 4

Methodology

4.1 Experimental Design

This project consist of two phases. In the first phase we apply feature selection techniques in the dataset in order to obtain the best features to make a classificator. Then we train and test with the machine learning algorithms: SVM, KNN with 1 and 10 neighbours, Ripper and Decision Trees. From all of them, we compare the results and select the one that generates the bests results detecting attacks. After that, we create the software INsIDES with the Java programming language and the API of Weka integrating the classificator that we selected before and the Argus software to perform the feature extraction. Figure 4.1 shows the general design which illustrates the whole procedure of the phase one.

In the second phase, we inject all the raw packet data of the UNSW-NB15 dataset to the network interface using the tool TcpReplay. From one side, we process the packet data with INsIDES and log the results to a MySQL database. On the other side, we process all this packets with Snort and log the results to the same MySQL database using the tool Barnyard2. Finally, we compare the results generated with both IDSs using the MySQL database and some scripts to automate the task. Figure 4.2 shows the general design which illustrates the whole procedure of the phase two.

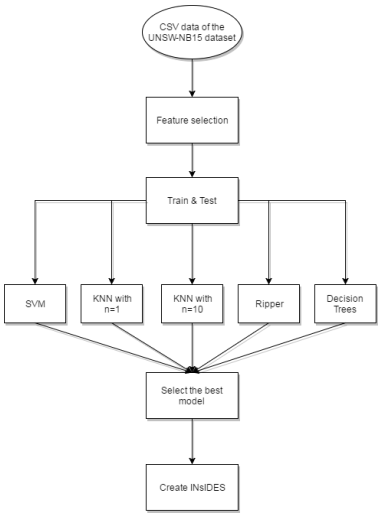


Figure 4.1: Project phase one design.

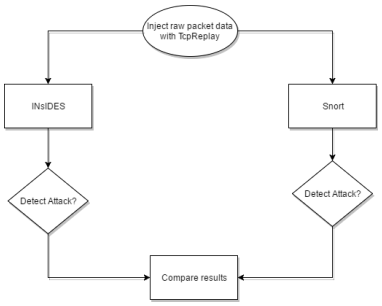


Figure 4.2: Project phase two design.

4.2 Feature selection

As we could see before, the portion of the UNSW-NB15 dataset has 45 features. Before executing any feature selection technique, we perform some changes to the training and testing datasets. We remove the features named *id* and *label* because the *id* of the flow data does not reflect any valuable information for the classification and the *label* only indicates if

it is an attack or not and we want to know the category of the attack.

Furthermore, we also removed the features that are not related to the network flow data and were extracted with the Bro tool. We make this important change because we want to prove that taking into account only the network flow data[21] we can make an IDS solution that can performs well. Additionally, making these changes we are able to make our solution a real time IDS solution because only depends of the Argus software to extract the features. Otherwise we would have to extract the features with both Argus and Bro, match the features with the network connections and then classify the traffic which involves a lot of processing time.

Hence, removing these features, our training and testing dataset now have 33 features including the class label *attack_cat*. In table 4.1 it can be seen the modified dataset.

Id	**Name**
1	dur
2	spkts
3	dpkts
4	sbytes
5	dbytes
6	rate
7	sttl
8	dttl
9	sload
10	dload
11	sloss
12	dloss
13	sinpkt
14	dinpkt
15	sjit
16	djit
17	swin

18	stcpb
19	dtcpb
20	dwin
21	tcprtt
22	synack
23	ackdat
24	smean
25	dmean
26	ct_state_ttl
27	ct_dst_ltm
28	ct_src_dport_ltm
29	ct_dst_sport_ltm
30	ct_dst_src_ltm
31	ct_src_ltm
32	is_sm_ips_ports
33	attack_cat

Table 4.1: Features used of the UNSW-NB15 dataset portion.

With these features we perform a feature selection technique called *InfoGainAttributeEval* with the search method *Ranker* using 10 fold cross-validation with 1 seed in Weka in order to select the best features: that ones that gives more information and gives the best results in the classification. Subsequently, we calculate the learning curve of the precision testing different ranges of attributes through the machine learning algorithms in order to reduce the training time and the possible overfitting reducing the feature set.

4.3 Feature extraction

To extract the selected features from the raw packet data we have integrated Argus within INsIDES. Using the Argus server we listen to the traffic in the network interface and redirect the output to a local port with the following command:

argus -i enp0s3 -P 4444

Where *enp0s3* is the network interface and 4444 is the local port to which we redirect the traffic so that the Argus client reads it. Then with the Argus client, we listen to the output of the Argus server in the same local port and extract only the selected features with the following command:

ra -S localhost:4444 -n -L -1 -c , -u -s feature1 featureN

This command connect with the Argus server in the localhost machine and extract all the features from 1 to N. For further information about the command options used check the following documentation pages of Argus server[22] and client[23].

However, all the features can not be extracted directly from Argus. The features *ct_dst_sport_ltm, ct_state_ttl, ct_dst_src_ltm, ct_src_dport_ltm* and *ct_dst_ltm* need some extra algorithms based in other features of the dataset. All of them can be easily programmed showing the paper related to the dataset except the *ct_state_ttl* feature.

The ct_state_ttl feature has to be calculated with the following pseudocode that was obtained directly from the authors of the dataset.

```
Input<-[dr<-data records, ct_state_ttl<-0]
for each record in dr do
  if (ds.sttl=(62||63||254||255) &&
        ds.dttl=(252||253) &&
        ds.State=FIN
        )then ct_state_ttl=1
  else if (ds.sttl=(0||62||254) &&
         ds.dttl= (0) &&
         ds.State=INT
        ) then ct_state_ttl=2
  else if (ds.sttl=(62||254) &&
        ds.dttl=(60||252||253)&&
        ds. State= CON
        ) then ct_state_ttl=3
```

```
    else if (ds.sttl = 254 &&
          ds.dttl= 252 &&
          ds.State=ACC
          ) then ct_state_ttl=4
    else if (ds.sttl = 254 &&
          ds.dttl= 252 &&
          ds.State=CLO
          ) then ct_state_ttl=5
    else if (ds.sttl = 254 &&
          ds.dttl= 0 &&
          ds.State= REQ
          ) then ct_state_ttl=7
    else ct_state_ttl=0
    end if
  end for
  Output->[ds.id, ct_state_ttl]
```

Listing 4.1: ct_state_ttl pseudocode

4.4 Generation of the multiclass classificators

To generate the multiclass classficators we execute the following machine learning algorithms with Weka using the training and testing dataset:

- Support Vector Machines: for this classification algorithm we used the SMO algorithm with the calibrator *Logistic* and the kernel *PolyKernel* with exponent 1, both are the default values of Weka.
- K-Nearest Neighbors: for this classificator we used the IBk algorithm with two values of k: 1 and 10 and the *LinearNNSearch* algorithm for search the neighbours.
- Ripper: for this classificator we used the JRip algorithm with the Weka default values.

- Decision Trees: for this classificator we used the J48 algorithm with the Weka default values.

4.5 Design and implementation of INsIDES

INsIDES is made using the Java programming language. As external software, it uses the API of Weka and the Argus server and client. Figure 4.3 shows the logical design followed to make the software. First of all, it

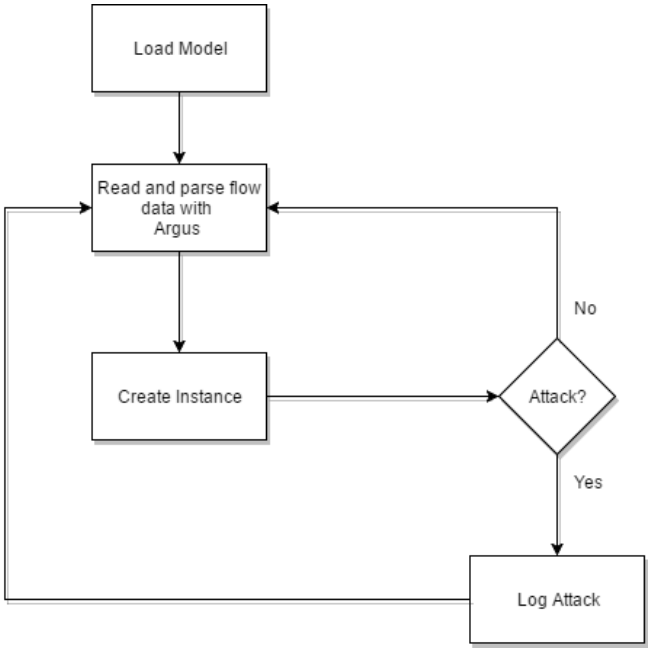


Figure 4.3: INsIDES logical design.

loads the selected classificator model in memory which we will use to test the instances with Weka. Then, it executes the Argus server to go sniffing the packets that arrive from the network interface. At the same time, it executes the Argus client in order to get the formatted features of the network flow data. For each network flow, the program creates an

instance with the features extracted as the attributes of the instance. After that, it test this instance through the classificator. If the classificator indicates that this record is an attack, it prints the record in csv format logging the following fields: *ip_src, src_port, ip_dst, dst_port, proto, class_name*. Finally, we load the csv file into a database table in order to perform the comparison with Snort.

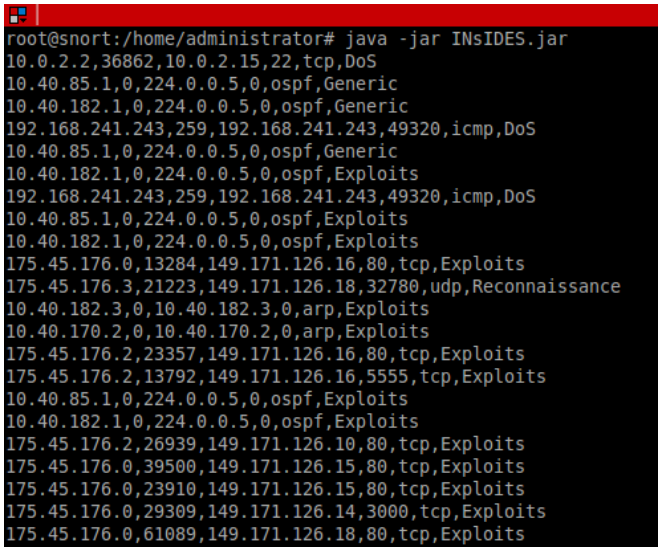


Figure 4.4: Example output of INsIDES

Figure 4.4 shows an example of the output of INsIDES running while detect attacks. Since Argus reads the network flow data directly from the network interface and Argus has been integrated into INsIDES, INsIDES is capable to perform a real time detection.

Chapter 5

Results

5.1 Feature selection result

Executing the InfoGainAttributeEval in Weka as stated before in the methodology chapter we obtain that the best features are, in order of relevance, those that can be seen in figure 5.1.

```
=== Attribute selection 10 fold cross-validation (stratified), seed: 1 ===

average merit      average rank  attribute
 1.637 +- 0.002       1   +- 0        4 sbytes
 1.332 +- 0.001       2   +- 0       24 smean
 1.244 +- 0.001       3   +- 0        9 sload
 0.918 +- 0.001       4   +- 0        5 dbytes
 0.789 +- 0.001       5   +- 0       25 dmean
 0.752 +- 0.001       6.1 +- 0.3      6 rate
 0.75  +- 0.001       6.9 +- 0.3     29 ct_dst_sport_ltm
 0.726 +- 0.001       8   +- 0        1 dur
 0.707 +- 0.001       9   +- 0       26 ct_state_ttl
 0.705 +- 0.001      10   +- 0        8 dttl
 0.695 +- 0          11   +- 0       30 ct_dst_src_ltm
 0.674 +- 0.001      12.2 +- 0.4      7 sttl
 0.674 +- 0          12.8 +- 0.4     28 ct_src_dport_ltm
 0.671 +- 0.001      14   +- 0        3 dpkts
 0.663 +- 0.001      15   +- 0       10 dload
 0.659 +- 0.001      16   +- 0       27 ct_dst_ltm
 0.646 +- 0.001      17   +- 0       14 dinpkt
 0.585 +- 0.002      18   +- 0       13 sinpkt
 0.563 +- 0.001      19   +- 0       31 ct_src_ltm
 0.544 +- 0.001      20   +- 0       22 synack
 0.542 +- 0.001      21.3 +- 0.46    21 tcprtt
 0.542 +- 0.001      21.7 +- 0.46     2 spkts
 0.528 +- 0.001      23   +- 0       23 ackdat
 0.515 +- 0.002      24   +- 0       15 sjit
 0.499 +- 0.001      25.1 +- 0.3     12 dloss
 0.498 +- 0.001      25.9 +- 0.3     16 djit
 0.44  +- 0.001      27   +- 0       11 sloss
 0.276 +- 0          28   +- 0       17 swin
 0.27  +- 0.001      29   +- 0       18 stcpb
 0.269 +- 0.001      30   +- 0       19 dtcpb
 0.269 +- 0.001      31   +- 0       20 dwin
 0.026 +- 0          32   +- 0       32 is_sm_ips_ports
```

Figure 5.1: InfoGainAttributeEval results with 10 fold cross-validation and 1 seed.

After that, we calculate the learning curve of the precision testing different ranges of attributes through the five machine learning algorithms in order to reduce feature set.

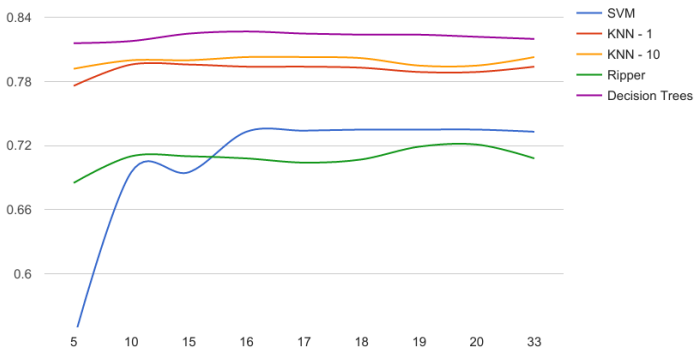


Figure 5.2: Precision learning curve with the five algorithms.

We can see in figure 5.2 that the maximum of precision is achieved with the Decision Trees algorithm inside the range of attributes 15 and 17. Hence we decided to select the first 16 features plus the class label, so finally, in the table 5.1 we have the 17 features selected.

Id	Name	Description
4	sbytes	Source to destination bytes
24	smean	Mean of the flow packet size transmitted by the source ip
9	sload	Source bits per second
5	dbytes	Destination to source bytes
25	dmean	Mean of the flow packet size transmitted by the destination ip

6	rate	Packets per second
29	ct_dst_sport_ltm	No of connections of the same destination ip and source port in 100 connections.
1	dur	Record total duration
30	ct_state_ttl	No. for each state according to specific range of values for source/destination time to live
8	dttl	Destination to source time to live
30	ct_dst_src_ltm	No. of connections of the same source and destination address in 100 connections
7	sttl	Source to destination time to live
28	ct_src_dport_ltm	No. of connections of the same source address and destination port in 100 connections
3	dpkts	Destination to source packet count
10	dload	Destination bits per second
27	ct_dst_ltm	No. of connections of the same destination address in 100 connections
33	attack_cat	The name of each attack category.

Table 5.1: Selected features for classification.

5.2 Multiclass classificators results

In order to take the best classificator for our software, we compare the Precision, Recall and F-Measure of each classificator.

Figure 5.3 compares the Precision, Recall and F-Measure between all the five algorithms. It can be seen clearly that the Decision Trees classificator yields the best results, which are 0.827 for Precision, 0.777 for Recall and 0.781 for F-Measure. Hence, the Decision Trees is the classificator selected to be integrated into INsIDES.

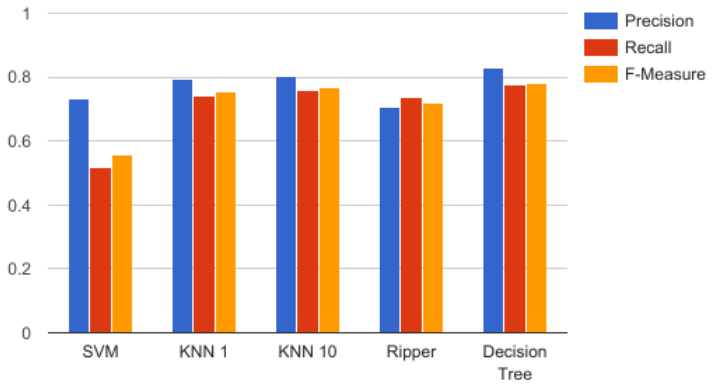


Figure 5.3: Classificators results

5.3 INsIDES and Snort binary classification results

To test the performance of INsIDES and Snort detecting attacks, we inject the full UNSW-NB15 dataset raw packet data through the network interface while we execute both programs and log the results. Figure 5.4 and figure 5.5 show the resulting confusion matrix tables of INsIDES and Snort respectively taking into account only if it is attack or not.

		Predicted INsIDES	
	2540047	Attack	No Attack
Actual Class	Attack	315211	6072
	No Attack	190057	2028707

Figure 5.4: Binary confusion matrix of INsIDES

Conventionally, to test the performance of an IDS software is calculated the accuracy, precision, detection rate and false alarm rate. We use

		Predicted SNORT	
	2540047	Attack	No Attack
Actual Class	Attack	7808	313475
	No Attack	680365	1538399

Figure 5.5: Binary confusion matrix of Snort

both confusion matrixs to make the calculations.

5.3.1 Accuracy

Accuracy can be defined as a percentage of correctly classified instances over the whole dataset. This research calculated accuracy rate based on the following formula:

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} \tag{5.1}$$

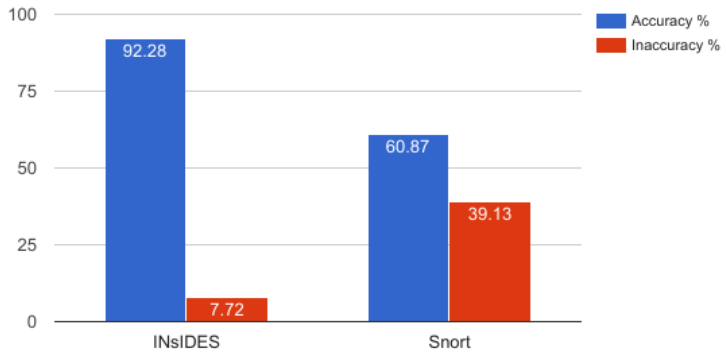


Figure 5.6: Accuracy results

Figure 5.6 compares the accuracy between INsIDES and Snort. The results indicate that INsIDES classifies correctly more instances than Snort. This means that the likelihood that Snort will miss an attack or even detect a normal traffic as if it was an attack is higher than INsIDES with the terrible consequences that this mistake could cause.

5.3.2 Precision

Precision can be defined as a percentage of correctly classified instances over the returned instances. This research calculated precision rate based on the following formula:

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

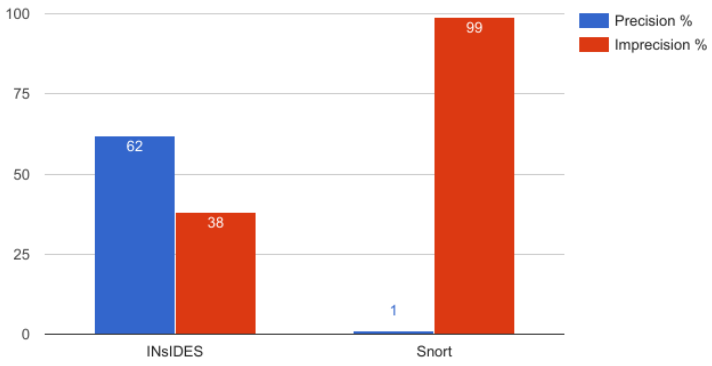


Figure 5.7: Precision result

Figure 5.7 compares the precision between INsIDES and Snort. The results show that INsIDES has more precision than Snort. This is due to the fact that Snort has a few number of TP instances which means that

Snort really does not detect correctly any attack of the dataset with the updated rules that we have downloaded and installed into it.

5.3.3 Detection Rate

The proportion of detected attacks over all the attacks is called Detection rate. This research calculated detection rate based on the following formula:

$$Detection\ rate = \frac{TP}{TP + FN} \tag{5.3}$$

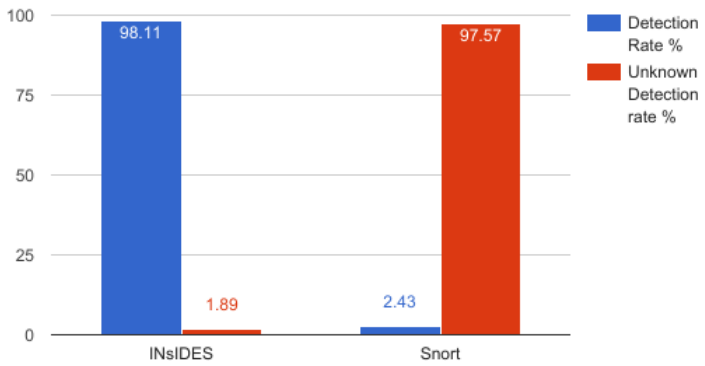


Figure 5.8: Detection rate results

Figure 5.8 compares the detection rate between INsIDES and Snort. It can be seen that exist a big difference between both softwares which means that Snort lose almost all the attacks whereas INsIDES is capable of detect correctly the 98.11% of them as an attack.

5.3.4 False Alarm Rate

False alarm rate indicates the amount of normal data which is falsely detected as an attack. This research calculated false alarm rate based on the following formula:

$$False\ alarm\ rate = \frac{FP}{FP + TN} \tag{5.4}$$

Based on the results, figure 5.9 had been generated which compares the average of the false alarm rate between both softwares.

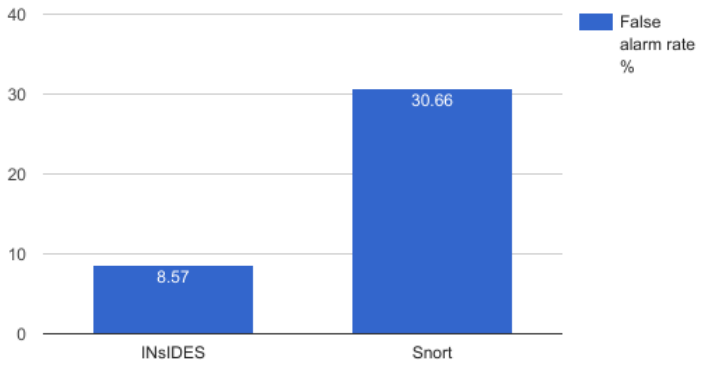


Figure 5.9: False alarm rate result

It can be seen clearly from figure 5.9 that false alarm rate of Snort is bigger than INsIDES which means that Snort gives more false positives adding so much noise to the system administration.

5.4 INsIDES multiclass classification result

We have showed before that INsIDES performs very well with binary classification. However, we want to know how it performs detecting each

class. For this purpose, we have calculated the confusion matrix of the figure 5.10. In this case, we didn't generated the confusion matrix for each class label of Snort because the classes of the dataset and the signatures used by Snort to label the attacks are different.

		Predicted INsIDES									
	2540047	Normal	Fuzzers	Analysis	Backdoor	DoS	Exploits	Generic	Reconnaissance	Shellcode	Worms
Actual Class	Normal	2028707	16246	911	607	3957	159153	8363	548	253	19
	Fuzzers	4546	15672	17	44	177	3419	48	152	161	10
	Analysis	302	53	204	2	55	2041	0	0	20	0
	Backdoors	31	130	0	214	107	1789	1	14	43	0
	DoS	317	394	9	24	2517	12752	37	97	202	4
	Exploits	619	294	56	125	592	41713	80	683	329	34
	Generic	83	261	4	84	428	2994	211489	29	95	14
	Reconnaissance	154	221	0	37	135	7085	9	6263	83	0
	Shellcode	20	78	0	9	31	142	18	57	1156	0
	Worms	0	1	0	1	2	30	4	1	0	135

Figure 5.10: Confusion Matrix for the multiclass classification

Using the confusion matrix we calculated the accuracy, precision, detection rate and false alarm rate for each class. The results can be shown in the figure 5.11.

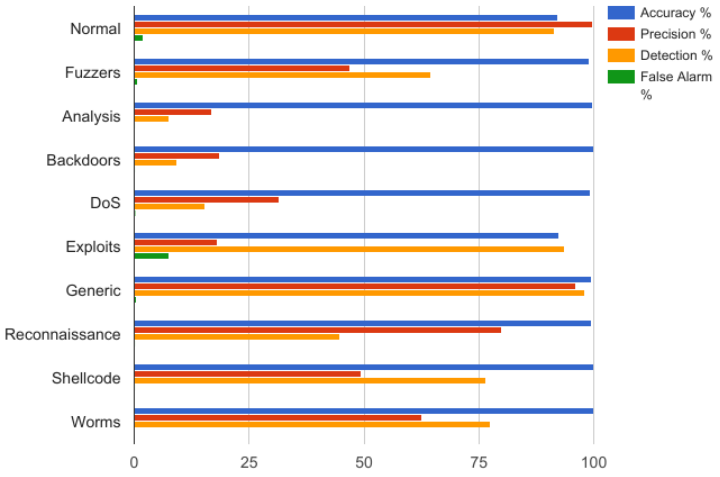


Figure 5.11: Accuracy, Precision, Detection Rate and False Alarm Rate for each class.

With these results we can observe that with our software we detect correctly nearly the 100% of instances in terms of accuracy which means that INsIDES normally classifies correctly the network events retrieved. However, we can see that in terms of detection rate we are losing attacks in the Analysis, Backdoors, DoS and Reconnaissance classes because we have this measure under the 50% in all classes. Also is not surprising that almost the same classes have a precision rate under the 50%. Finally, we notice that the overall false alarm rate is low.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this project, we presented how we created a new machine learning-based IDS software that is capable to classify malicious and benign traffic and attribute malicious activity to an attack family in real time. According to investigation of recent works that are performed on UNSW-NB15 dataset, this project is the first proposed work to generate an IDS solution from it. Additionally, we have also proved that taking into account only the features related to the network flow data we can get an optimal IDS being able to achieve the detection in real time. Finally, the results of INsIDES show promising development in the application of machine learning techniques for intrusion detection systems. It has been proven that the results from our system are better than those shown by rule based approaches like Snort although the rules were up to date at the moment of the project. Hence, we have proved that the rule based systems nowadays are outdated and we need to move towards a more effective solution like INsIDES.

6.2 Future Work

In future work, we plan to implement in INsIDES the ability to learn how to detect new types of attacks. Additionally, we plan to study why the selected features are relevant to the intrusion detection task. Moreover, INsIDES does not filter any malicious connection, it only log the attacks. Another improvement can be include the IPS functionality into INsIDES in order to stop the attacks in real time. Furthermore, we plan to test INsIDES with other available IDS software like Suricata or Bro-IDS to know how will perform our software compared with other available IDS solutions. Also, we intend to extend the project toward transfer learning techniques to improve detection from untrained network environments like a real traffic of a company or institution.

Bibliography

[1] S. M. Hussein, F. H. M. Ali, and Z. Kasiran, “Evaluation effectiveness of hybrid IDS using snort with Na??ve Bayes to detect attacks,” *2012 2nd International Conference on Digital Information and Communication Technology and its Applications, DICTAP 2012*, pp. 256–260, 2012.

[2] Wikipedia, “Computer Security,” 2017. [Online]. Available: https://en.wikipedia.org/wiki/Computer{_}security

[3] Symantec, “Internet Security Threat Report,” Tech. Rep., 2017. [Online]. Available: https://digitalhubshare.symantec.com/content/dam/Atlantis/campaigns-and-launches/FY17/ThreatProtection/ISTR22{_}Main-FINAL-APR24.pdf?aid=elq{_}

[4] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, “An Overview of IP Flow-Based Intrusion Detection,” *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 343–356, 2010.

[5] Wikipedia, “Defense in Depth.” [Online]. Available: https://en.wikipedia.org/wiki/Defense{_}in{_}depth{_}(computing)

[6] Y. Hamid, M. Sugumaran, and V. Balasaraswathi, “IDS Using Machine Learning - Current State of Art and Future Directions,” *British Journal of Applied Science & Technology*, vol. 15, no. 3, pp. 1–22, 2016.

[7] C. F. Tsai, Y. F. Hsu, C. Y. Lin, and W. Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009.

[8] A. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. PP, no. 99, p. 1, 2015.

[9] Y. X. Ding, M. Xiao, and A. W. Liu, "Research and implementation on snort-based hybrid intrusion detection system," *Proceedings of the 2009 International Conference on Machine Learning and Cybernetics*, vol. 3, no. July, pp. 1414–1418, 2009.

[10] C. Science and K. Mangalore, "A Two-tier Network based Intrusion Detection System Architecture using Machine Learning Approach," pp. 42–47, 2016.

[11] N. Fallahi and A. Sami, "Automated Flow-based Rule Generation for Network Intrusion Detection Systems," pp. 1948–1953, 2016.

[12] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*, 2009.

[13] J. McHugh, "Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory," *ACM Transactions on Information and System Security*, vol. 3, no. 4, pp. 262–294, 2000.

[14] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," *2015 Military Communications and Information Systems Conference (MilCIS)*, pp. 1–6, 2015.

[15] ——, "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison

with the KDD99 data set,” *Information Security Journal: A Global Perspective*, vol. 25, no. 1-3, pp. 18–31, 2016.

[16] Weka, “Weka.” [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/

[17] Argus, “Argus.” [Online]. Available: https://qosient.com/argus/

[18] TcpReplay, “TcpReplay.” [Online]. Available: http://tcpreplay.appneta.com/

[19] Snort, “Snort.” [Online]. Available: https://www.snort.org/faq/what-is-snort

[20] ForensicsWiki, “Barnyard2.” [Online]. Available: http://forensicswiki.org/wiki/Barnyard2

[21] A. Sperotto and A. Pras, “Flow-based intrusion detection,” in *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management, IM 2011*. IEEE, may 2011, pp. 958–963.

[22] QOSIENT, “Argus Server Documentation.” [Online]. Available: http://qosient.com/argus/man/man8/argus.8.pdf

[23] ——, “Argus Client Documentation.” [Online]. Available: http://qosient.com/argus/man/man1/ra.1.pdf