Anomaly-based network intrusion detection using machine learning

Maxime Labonne

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2020IPPAS011

Thèse de doctorat

TELECOM SudParis

IP PARIS

Anomaly-Based Network Intrusion Detection Using Machine Learning

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom SudParis

École doctorale n°580 Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat : sécurité informatique et intelligence artificielle

Thèse présentée et soutenue à Palaiseau, le 05/10/2020, par

MAXIME LABONNE

Composition du Jury :

Joaquin Garcia-Alfaro Professeur, Télécom SudParis	Président
Steven Martin Professeur, Université Paris-Sud (LRI)	Rapporteur
Bruno Volckaert Professeur, Ghent University (IBCN)	Rapporteur
Jean-Philippe Fauvelle Ingénieur de recherche, Airbus Defense & Space	Examinateur
Djamal Zeghlache Professeur, Télécom SudParis	Directeur de thèse
Alexis Olivereau Ingénieur de recherche, CEA LIST (LSC)	Co-directeur de thèse

Acknowledgments

First of all, I would like to thank my thesis co-supervisor Alexis Olivereau, who gave me the chance to do this PhD. His constant good humour and sharp witty traits brightened up our long and numerous thesis meetings. His expertise in the field has guided me throughout these three years, and his sustained confidence has been a real source of motivation for me in my work.

I would like to thank my thesis supervisor Djamal Zeghlache for his sound advice, both academically and professionally. He has helped me to gain insight into my own work and provided new perspectives that have led me to be more creative in my approaches. His unfailing support has pushed me to do my best to live up to his expectations.

I would like to thank the members of my thesis committee. Many thanks to Prof. Bruno Volckaert from Ghent University and Prof. Steven Martin from Université Paris-Sud for their valuable time in reviewing this manuscript as evaluators. I am grateful for their comments and questions that helped me improve this document. I would also like to thank Prof. Joaquin Garcia-Alfaro from Télécom SudParis and Mr. Jean-Philippe Fauvelle from Airbus Defence Space for their precious work as examiners.

I would like to thank all my colleagues in the LSC team for their friendship and discussions. I will remember our technical conversations and our less technical debates over a cup of coffee. I feel lucky and honored to have been able to work in such a good environment.

Last but not least, I would like to thank my parents, my sister Pauline, and my brother Alix for their support throughout this journey.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

AE: AutoEncoder
AIS: Artificial Immune System
ANN: Artificial Neural Network
API: Application Programming Interface
ARIMA: AutoRegressive Integrated Moving Average
ARP: Address Resolution Protocol
ART: Adaptive Resonance Theory
AUC: Area Under the Curve
AUROC: Area Under the Curve of the Receiver Operating Characteristic
AWS: Amazon Web Service
BiLSTM: Bidirectional Long Short-Term Memory
CIA: Confidentiality Integrity Availability
CIC: Canadian Institute for Cybersecurity
CNN: Convolutional Neural Network
CPU: Central Processing Unit
CSE: Communications Security Establishment
DBN: Deep Belief Network
DNS: Domain Name System
DoS: Denial of Service
DDoS: Distributed Denial of Service
ENN: Edited Nearest Neighbours
FN: False Negative
FP: False Positive
FPR: False Positive Rate
FTP: File Transfer Protocol
GAN: Generative Adversarial Network
GHSOM: Growing Hierarchical Self-Organizing Map
GPU: Graphics Processing Unit
HIDS: Host-based Intrusion Detection System
HTTP: Hypertext Transfer Protocol
ICA: Independent Component Analysis
ICMP: Internet Control Message Protocol
IDS: Intrusion Detection System

IDES: Intrusion Detection Expert System
IP: Internet Protocol
IPS: Intrusion Prevention System
ISO: International Organization for Standardization
ISP: Internet Service Provider
KDD: Knowledge Discovery and Data mining
KNN: K-Nearest Neighbors
LAN: Local Area Network
LSTM: Long Short-Term Memory
MAC: Media Access Control
MAE: Mean Absolute Error
MLP: MultiLayer Perceptron
MSE: Mean Squared Error
NIDS: Intrusion Detection System
NLP: Natural Language Processing
NS: Network Simulator
OS: Operating System
OVS: Open vSwitch
PCA: Principal Component Analysis
PSD: Power Spectral Density
QoS: Quality of Service
R2L: Remote To Local
RAID: Redundant Array of Independent Disks
RAM: Random-Access Memory
RBF: Radial Basis Function
RBFN: Radial Basis Function Network
RBM: Restricted Boltzmann Machine
ReLU: Rectifier Linear Unit
RFC: Request For Comments
RMSE: Root Mean Squared Error
RNN: Reccurent Neural Network
ROC: Receiver Operating Characteristic
SDN: Software-Defined Networking
SIEM: Security Information and Event Management
SMBO: Sequential Model-Based Optimization
SMO: Sequential Minimal Optimization
SMOTE: Synthetic Minority Over-sampling Technique
SNMP: Simple Network Management Protocol
SOM: Self-Organizing Map
SQL: Structured Query Language
SSH: Secure Shell
SVM: Support Vector Machine
TCP: Transmission Control Protocol

TN: True Negative
TP: True Positive
TPE: Tree-structured Parzen Estimator
TPR: True Positive Rate
U2R: User To Root
UDP: User Datagram Protocol
USAF: United States Air Forces
VLAN: Virtual Local Area Network
VM: Virtual Machine
XSS: Cross-Site Scripting

Chapter 1

Introduction

1.1 The Security Background

With the constant growth of the Internet, computer attacks are increasing not only in numbers but also in diversity: ransomware are on the rise like never before, and zero-day exploits become so critical that they are gaining media coverage. Antiviruses and firewalls are no longer sufficient to ensure the protection of a company network, which should be based on multiple layers of security. One of the most important layers, designed to protect its target against any potential attack through a continuous monitoring of the system, is provided by an Intrusion Detection System (IDS).

Current IDSs fall into two major categories: signature-based detection (or "misuse detection") and anomaly detection. For signature-based detection, the data monitored by the IDS is compared to known patterns of attacks. This method is very effective and reliable, widely popularized by tools like Snort [1] or Suricata [2], but has a major drawback: it can only detect known attacks that have already been described in a database. On the other hand, anomaly detection builds a model of normal behavior of the system and then looks for deviations in the monitored data. This approach can thus detect unknown attacks but often generates an overwhelming number of false alarms. During the past two decades, much research has been focused on anomaly-based IDSs. Indeed, their ability to detect unknown attacks is significant in a context where attacks are becoming more numerous and diverse.

Many machine learning techniques have been proposed for misuse and anomaly detection. These techniques rely on algorithms with the ability to learn directly from the data, without being explicitly programmed. This is particularly convenient considering the great diversity of the traffic. However, despite these advantages, anomaly detection algorithms are rarely deployed in the real world and misuse detection still prevails. The problem of the high false positive rate is often cited as the main reason for the lack of adoption of anomaly-based IDS [3]. Indeed, even a false positive rate of 1% can create so many false alarms on a high traffic network that they become impossible for an administrator to process.

The objective of this thesis is to propose solutions to improve the quality of detec-

tion of anomaly-based IDS using machine learning techniques for deployment on real networks. Improving the accuracy of detection on known datasets is not enough to achieve this goal, because the results obtained are not transferable to real networks. Indeed, machine learning models learn the traffic of a dataset and not the traffic to be monitored. They need to be re-trained on the monitored network, which is hardly possible as it requires labeled datasets containing attacks on a real network. The second objective of the thesis is therefore to develop IDSs that can be deployed on unknown networks without labeled datasets.

1.2 Contributions

The main contributions of this thesis are as follows:

- We conducted a survey on the state of the art of neural network classifiers for intrusion detection on KDD Cup 99 and NSL-KDD [4]. We surveyed more than 70 papers on this topic from 2009 to 2017 to identify areas where improvements can be made and which neural network architectures are the most efficient.
- We proposed an efficient architecture for intrusion detection on KDD Cup 99 and NSL-KDD using machine learning models [5] [6]. This architecure is based on a three-step optimization method: 1/ data augmentation; 2/ parameters optimization; and 3/ ensemble learning. This approach achieved a very high classification accuracy (94.44% on KDD Cup 99 test set and 88.39% on NSL-KDD test set) with a low false positive rate (0.33% and 1.94% respectively).
- We introduced a Snort signature generator from anomalies, which automates the signature creation process and thus speeds up the update of misuse-based IDS databases. It can also be used within a hybrid IDS to self-populate its own signature database.
- We studied the capacities of transfer learning to solve the problem of the lack of labelled datasets on real networks. We show that transfer learning is relevant for certain types of attacks (brute-force).
- We patented a method and system for detecting anomalies in a telecommunications network based on the individual analysis of protocol headers [7]. This anomaly detection method uses ensemble learning to assign each monitored packet an anomaly score. This unsupervised learning method is our solution to solve the problem of lack of datasets on real networks.
- We applied this method to a recent and realistic dataset (CICIDS2017) over a 4-day period to prove its effectiveness [8]. This approach successfully detects 7 out of 11 attacks not seen during the training phase, without any false positives.
- We proposed a solution to predict the bandwidth utilization between different network links with a very high accuracy [9]. A simulated network is created to

collect data related to the performance of the network links on every interface. Our model's predictions of bandwidth usage in 15 seconds rarely exceed an error rate of 3%.

1.3 Thesis Organization

The remainder of the thesis is organized as follows:

Chapter 2 introduces concepts essential to intrusion detection. It defines what a computer attack is, what an intrusion detection system is, and provides a historical perspective of the field. Different types of IDSs are detailed, with their strengths and weaknesses. The contributions of machine learning in this field are explained and the different specific datasets discussed in this dissertation are presented. Finally, the most commonly used metrics are defined.

Chapter 3 presents existing work related to intrusion detection using machine learning algorithms. The different models are detailed with a short presentation of how they work. Approaches and results are successively presented, then compared in a common section to determine the best techniques. Finally, the problems identified are listed along with ideas on how to improve these different points. The insights gathered in this chapter are used in the design of the IDSs in the following chapters.

Chapter 4 presents two solutions using learning machine models to classify attacks on the two most popular datasets in intrusion detection. Data augmentation is used to rebalance these datasets and to improve detection of the rarest attacks. Different models are then trained and optimized to obtain the best quality of detection. Finally, they are combined using a specific rule to improve their accuracy.

Chapter 5 describes two methods to improve two aspects of intrusion detection. Firstly, it is possible to improve the update of signature databases of misuse-based IDS by generating these signatures from anomalies. A hybrid IDS could then self-populate its own signature database. Secondly, networks where IDSs are deployed rarely provide labeled datasets containing attacks. Transfer learning is studied to train models on labeled datasets and then transfer these models to real-life networks that do not contain attacks.

Chapter 6 presents a method of intrusion detection without the need for a labelled dataset (unsupervised learning). This technique performs anomaly detection by learning the behavior of the protocol headers of the monitored network. The scores obtained by the different protocols in a single packet are aggregated to produce the packet anomaly score. A succession of abnormal packets is considered as an indicator of an attack.

Chapter 7 focuses on denial of service attacks, and more generally on network congestion problems. Models are trained to predict the bandwidth consumption between different links in a simulated network. This method works in real time in combination with Software-Defined Networking (SDN), allowing congestion problems to be corrected before they occur.

Chapter 8 concludes the thesis by summarizing the main points of the dissertation. The relevance of machine learning for intrusion detection and future work are discussed.

Chapter 2

Concepts and background

2.1 Intrusion Detection System (IDS)

2.1.1 Definition

Confidentiality, integrity, and availability (also known as the CIA triad) are three fundamental concepts of information security. A cyber attack (or an intrusion) is defined as all unauthorized activities that compromise one, two, or all of these three components of an information system.

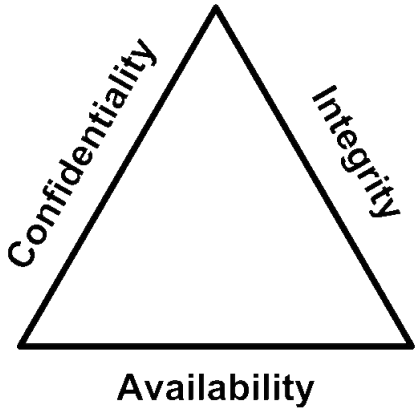

Figure 2.1: CIA triad.

- Confidentiality is defined by the ISO 27000 standard as the "property that information is not made available or disclosed to unauthorized individuals, entities, or processes" [10]. This information may include personal data, credit card numbers, or more generally any information considered private. The challenge of confidentiality is to allow legitimate users to access this information while preventing

others from doing so. A failure of confidentiality results in a data breach that cannot be remedied, but can be managed in a way that minimizes its impact on users. Confidentiality is implemented using different security mechanisms like encryption, passwords, two-factor authentication, security tokens, etc. The level of confidentiality of the information is correlated with the strength of the associated security measures. The same information can thus be protected by several layers of protection, combining authentication mechanisms and cryptography.

- Integrity is defined by the ISO 27000 standard as the "property of accuracy and completeness" [10]. Integrity ensures that information is protected from being modified by unauthorized parties, or accidentally by authorized parties. An integrity problem can, for example, allow the amount of online transactions to be modified. A malicious attacker can also insert himself into a conversation between two parties to impersonate one of them. Then, the attacker can gain access to information that the two parties were trying to send to each other (man-in-the-middle attack). Integrity is commonly implemented using encryption, version control, checksums, and hashing. The received data is hashed and compared to the hash of the original data. Moreover, information can be changed by non-human-caused events such as server crash or electromagnetic pulse. Redundant systems and backup procedures are important security mechanisms to ensure data integrity.

- Availability is defined by the ISO 27000 standard as the "property of being accessible and usable on demand by an authorized entity" [10]. Unavailability of information can have serious consequences. Denial of Service (DoS) attacks are common attacks against availability. For example, attackers may bring down servers and make services unavailable to legitimate users by flooding the targeted machines with superfluous requests. This illegitimate traffic is often detected and blocked by security mechanisms such as firewalls and Intrusion Detection Systems (IDSs). Power outages and natural disasters like flood or fire can also lead to lack of availability. A disaster recovery plan is required to minimize the impact of these disasters, including redundancy, failover, RAID and off-site backups.

Monitoring network traffic and computer events to detect malicious or unauthorized activities is a process called "intrusion detection". Every device or software application whose goal is to conduct an intrusion detection is considered as an Intrusion Detection System (IDS). Figure 2.2 shows how an IDS transforms monitored activities into alerts using its knowledge (database, statistics, artificial intelligence, etc.). These alarms are then reported either to an administrator or collected centrally using a Security Information and Event Management (SIEM) system. A SIEM system provides real-time analysis of the outputs of multiple sources to correlate the different alerts and show a comprehensive view of IT security.

IDSs are sometimes confused with two other security tools: firewalls and Intrusion Prevention Systems (IPSs). These three security mechanisms are designed to protect systems within a network but use different means. For instance, firewalls look outwardly

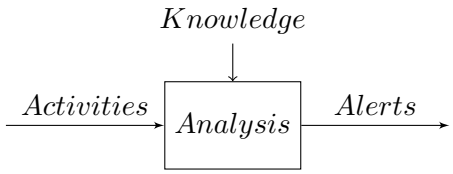

Figure 2.2: Intrusion Detection System function.

for intrusions in order to stop them before they enter the protected network. They analyze packet headers to filter incoming and outgoing traffic based on predetermined rules (protocol, IP address, port number...). On the other hand, IDSs are able to monitor activities within the protected network and not just at its perimeter. Unlike a firewall, IDSs only have a monitoring role: they cannot take action to block suspicious activities and therefore need an administrator to process their alerts. This is not the case with IPSs, which function as an IDS but are able to proactively block a detected threat. This automation adds a layer of complexity since an inappropriate response can cause additional problems on the network.

During the 1970s, the growth of computer networks created new problems related to the monitoring of user activities and access. In October 1972, the United States Air Forces (USAF) published a paper written by James P. Anderson, outlining the fact that the USAF "become increasingly aware of computer security problems. This problem was felt virtually in every aspect of USAF operations and administration." [11] The particular challenge of USAF was due to the fact that users with different levels of security clearance shared the same computer systems.

The same author published another paper in 1980, detailing several methods to improve computer security threat monitoring and surveillance [12]. James P. Anderson introduces the idea of automating the detection of intrusion within a network in order to detect "clandestine" users. The goal of this IDS was to help administrators to review system event logs, file access logs, and user access logs. The concept of an intrusion detection system is credited to this author for this paper.

Dorothy E. Denning published a model for real-time intrusion detection in 1986 [13]. This work was based on a prototype called Intrusion Detection Expert System (IDES), developed between 1984 and 1986. IDES uses a rule-based expert system to detect known attacks and statistical anomaly detection on user and network data. This system outputs several types of alerts, using a specific format in order to be system-independent.

2.1.2 Types of IDSs

IDSs can be classified into three categories according to the type of activities that are analyzed: network-based IDSs, host-based IDSs, and application-based IDSs.

- A Host-based IDS (HIDS) is an agent installed on individual hosts, which analyzes their activity: files, processes, system logs, etc. HIDSs have multiple resources at their disposal. Snapshots of the system can be compared to check for the presence

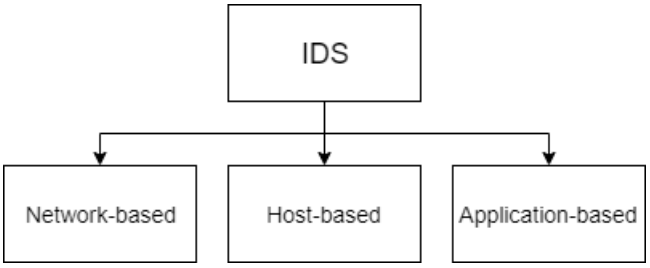

Figure 2.3: Classification of IDSs by analyzed activities.

of unauthorized or suspicious activity. Multiple failed login attempts, unusual high CPU usage for a long period of time are clues of potential attacks. Some HIDSs can also perform kernel-based detection by analyzing system calls and changes to system binaries. They also could be used to spy on the users by monitoring their activities.

- A Network-based IDS (NIDS) typically uses sensors at various points on the network. The analysis of the traffic is either accomplished by the sensor itself or remotely by a central controller. NIDSs are more scalable and cross-platform than HIDSs, this is why they are more widespread to protect a company's IT equipment. However, these solutions can be employed simultaneously to ensure a higher level of security. In this thesis, the term "IDS" always refers to NIDSs.
- An Application-based IDS is a special type of HIDS designed to monitor a specific application. Application-based IDSs analyze the interactions between users and applications: file executions or modifications, logs, authorizations, and any other potential abnormal activities. They can profile specific users to identify suspicious events. Some application-based IDSs can also access data before they are encrypted, acting as a middleware between the application and the encrypted data for storage.

IDSs can also be classified according to the detection method they use. They fall into three categories: signature-based detection, anomaly-based detection, and hybrid detection.

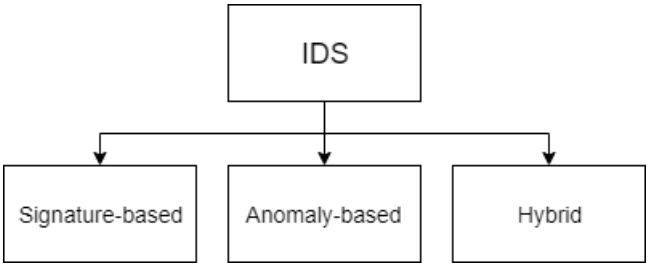

Figure 2.4: Classification of IDSs by detection method.

- Signature-based detection (also known as "misuse detection") comes with a database of known attack signatures. It compares monitored data with the signature database. A misuse detection IDS checks the input stream for the presence of an attack pattern like a classic antivirus. This signature can take the form of a sequence of bytes or characters, but more complex patterns are often represented as a branching tree diagram. To be efficient, the database of this kind of IDSs must be updated regularly. However, even with the latest updates, only known attacks can be detected using this method.

- Anomaly detection tries to learn a "normal" or "expected" behavior of the system. Any deviation from this behavior is considered as a potential attack and will generate an alarm. This method does not require updates or even the presence of a database. It can identify unknown attacks but also creates a lot of false positives that are difficult to process. It is also more difficult to collect information about the attack since it is not clearly identified by a signature.

- Hybrid detection combines the two solutions to mitigate weaknesses of each category: anomaly detection then misuse detection, misuse detection then anomaly detection, or both at the same time. The goal is to detect known attacks with their signatures, and to use anomaly detection to identify unknown intrusions.

2.2 Machine learning

2.2.1 Machine learning tasks

Developing a machine learning algorithm involves two specific steps: training and testing. Each model has its own training techniques. This process and the design of machine learning model is usually managed by a framework such as scikit-learn [14], Tensorflow [15], PyTorch [16], Matlab or Weka [17]. The framework has a strong impact on the optimization of the algorithm or the number of available parameters.

Machine learning models can perform many tasks, three of which are particularly interesting for intrusion detection: classification, regression, and reconstruction. Classification categorizes entries into several classes, such as "normal" or "attack", or even different families of attacks. Regression (also called "prediction") is used to determine continuous values, including a probability that an input is an attack. Finally, reconstruction is specific to a certain type of neural network. This task tries to reconstruct the input data by compressing and decompressing them to force the network to learn the features (representation learning).

Machine learning algorithms are trained in two different ways: in a supervised or unsupervised manner. Some models can be trained in both ways, such as neural networks. Most models use supervised training, where the dataset includes both inputs and the correct results associated with them. The algorithm learns to model the mathematical function that associates these results with the corresponding inputs. Classification and regression are two classic supervised training tasks. On the other hand, unsupervised

training does not use any results in its training dataset. Its purpose is to understand interesting structures within the input data. Reconstruction is an example of an unsupervised task.

Once the training is over, machine learning models need to be tested to assess their performance. This evaluation must involve new data, which were not a part of the training set. Otherwise, the evaluation would be biased because the model has already seen this data, and the correct results in case of supervised learning.

A validation set can also be used to compare different values of a parameter (for example, a learning rate or a number of neurons). After training, the value with the best results on the validation set is used. Then the whole network is tested on the test set. A validation set must also be composed of new data, often a small partition of the training set is reserved to this task.

2.2.2 Datasets

Large amount of data are required in order to train machine learning algorithms. The quality and quantity of data is crucial in any machine learning problem. Indeed, these problems are very data-dependent: more high-quality data can often beat better algorithms, which is why they are so important. Unfortunately, such datasets are also expensive and difficult to produce. In the intrusion detection field, two datasets are particularly popular despite some deficiencies: KDD Cup 99 and NSL-KDD. Two other more recent datasets were also used to address some of the shortcomings of the previous sets: CICIDS2017 and CSE-CIC-IDS2018.

Many other datasets exist for intrusion detection, as shown in this survey [18]. We have chosen to consider two old but popular datasets, and two recent datasets with many realistic attacks. Other datasets could have been chosen, such as Kyoto2006+ [19], Utwente [20], or UNSW-NB15 [21]. Datasets focused on a single type of attack (DoS, botnet...) were not considered.

2.2.2.1 KDD Cup 99

KDD CUP 99 has been one of the most popular dataset since its release in 1999. Developed by MIT Lincoln Labs, the objective was to advance a standard set of data with a wide variety of attacks to survey and evaluate research in intrusion detection. The dataset uses nine weeks of raw TCP dump data from a simulated U.S. Air Force network, with the addition of numerous attacks. Packets belonging to the same connection are merged into connection records. More specifically, the training data was processed into about five million connection records, collected in seven weeks, while the test data is composed of around two million connection records, collected in two weeks [22]. Training data is supposed to be used during the learning process of a machine learning technique, and test data on a fully trained solution to evaluate its performance.

Each connection is either labeled as normal or as one of four categories of attacks: Denial of Service (DoS), network probe (Probe), Remote to Local (R2L) and User to Root (U2R). A DoS attack is an attempt to make a machine or a service unavailable

Table 2.1: Examples of KDD CUP 99 features.

Feature name	Type	Description
duration	continuous	length (number of seconds) of the connection
protocol_type	discrete	type of the protocol, e.g. tcp, udp, etc.
service	discrete	network service on the destination, e.g., http, telnet, etc.
src_bytes	continuous	number of data bytes from source to destination
dst_bytes	continuous	number of data bytes from destination to source
flag	discrete	normal or error status of the connection
land	discrete	1 if connection is from/to the same host/port; 0 otherwise
wrong_fragment	continuous	number of "wrong" fragments
urgent	continuous	number of urgent packets
hot	continuous	number of "hot" indicator
num_failed_logins	continuous	number of failed login attempts
logged_in	discrete	1 if successfully logged in; 0 otherwise

for the users. A probe attack is a malignant network activity, such as port scanning, to learn about the architecture of the network. A R2L attack occurs when an attacker gains local access to a system via the network. A U2R attack exploits vulnerabilities in the system to gain super user privileges. Table 2.2 shows that this dataset is strongly imbalanced, with a lot more DoS attacks than U2R attacks for example. Test data and training data do not share the same probability distribution either.

Table 2.2: Distribution of KDD Cup 99 classes.

	Normal	DoS	Probe	R2L	U2R
Training	97278 (19.69%)	391458 (79.24%)	4107 (0.83%)	1126 (0.23%)	52 (0.01%)
Test	60593 (19.48%)	229855 (73.90%)	4166 (1.34%)	16345 (5.26%)	70 (0.02%)

These four attack categories can be further divided into 38 test attack types, and 24 training attack types. Each connection also possesses 41 derived features, as shown in Table 2.1. These features, numerical or symbolic, are analyzed by the classifier to distinguish normal connections from attacks. KDD Cup 99, although popular, suffers from several deficiencies as pointed out in analyses [23]. The dataset contains a huge number of redundant and duplicated records, and other mistakes that affect the performance of classifiers. They become biased towards more frequent records, whose numbers have been inflated.

2.2.2.2 NSL-KDD

A solution to most of these problems can be found in the NSL-KDD dataset [24]. This improved version of KDD Cup 99 removes redundant and duplicate records (about 78% and 75% of the records). It also reduces the number of connections in the train and test sets: from 805 050 total connections in KDD Cup 99 to 148 517 in NSL-KDD. Connections are also divided in different difficulty level groups. This categorization can help classifiers during their training phase to recognize the attacks that are the most difficult to detect. Nonetheless, NSL-KDD is not perfect and keeps certain inherent problems of KDD Cup 99. For example, attacks in this dataset are very old and do not represent what can be found in a modern network. There is also the issue of their synthetic origin, which cannot be corrected without redoing a dataset from scratch.

Table 2.3: Distribution of NSL-KDD classes.

	Normal	DoS	Probe	R2L	U2R
Training	67343 (53.46%)	45927 (36.46%)	11656 (9.25%)	995 (0.79%)	52 (0.04%)
Test	9711 (43.08%)	7460 (33.09%)	2421 (10.74%)	2885 (12.80%)	67 (0.30%)

However, these flaws are shared between the two datasets, which is why the use of NSL-KDD should be encouraged over KDD Cup 99.

2.2.2.3 CICIDS2017

CICIDS2017 is a dataset designed for IDSs and IPSs by the Canadian Institute for Cybersecurity. Unlike NSL-KDD, this dataset contains original data, unrelated to KDD Cup 99. The goal of the authors is to propose a reliable, publicly available IDS evaluation dataset on a realistic network, with a diverse set of modern attack scenarios. It was designed to solve the problem of lack of a up-to-date and credible dataset for intrusion detection.

CICIDS2017 provides 5 days of traffic, from Monday, July 3, 2017 to Friday July 7, 2017. The first day contains only normal traffic, while the 4 next days include normal traffic and 14 types of attacks: brute-force (FTP-Patator and SSH-Patator), Denial of Service (slowloris, SlowHTTPTest, Hulk, GoldenEye), Heartbleed, web attacks (brute-force, SQL injection, XSS), infiltration of the network from inside, botnet, Distributed Denial of Service (ARES), and port scanning. CICIDS2017 consists of 3 119 345 labeled network flows (83 features) and 56 329 679 network packets, captured with CICFlowMeter [25]. This dataset is also highly imbalanced, with 83.34% normal flows against 0.00039% flows labelled as “Heartbleed” [26].

Data are extracted from a simulation much closer to the behavior of a modern computer network. The attacks were created using real tools available online and credible strategies. Its large amount of data also helps to train deep neural networks, which

Table 2.4: Examples of CICIDS2017 features.

Feature name	Type	Description
Flow duration	continuous	duration of the flow in microsecond
total Fwd Packet	continuous	total packets in the forward direction
total Bwd packets	continuous	total packets in the backward direction
total Length of Fwd Packet	continuous	total size of packet in forward direction
total Length of Bwd Packet	continuous	total size of packet in backward direction
Fwd Packet Length Min	continuous	minimum size of packet in forward direction
Fwd Packet Length Max	continuous	maximum size of packet in forward direction
Fwd Packet Length Mean	continuous	mean size of packet in forward direction
Fwd Packet Length Std	continuous	standard deviation size of packet in forward direction
Bwd Packet Length Min	continuous	minimum size of packet in backward direction
Bwd Packet Length Max	continuous	maximum size of packet in backward direction
Bwd Packet Length Mean	continuous	mean size of packet in backward direction

require more information to be efficient . CICIDS2017 is a more reliable dataset than KDD Cup 99 and NSL-KDD for measuring the performance of an IDS.

2.2.2.4 CSE-CIC-IDS2018

CSE-CIC-IDS2018 is a collaborative project between the Communications Security Establishment and the Canadian Institute for Cybersecurity. It provides 10 days of traffic, from Wednesday, February 14, 2018 to Friday, March 2, 2018 with a focus on Amazon Web Services (AWS). This includes seven different attack scenarios that are similar to CICIDS2017: brute-force (FTP-Patator and SSH-Patator), Denial of Service (slowloris, SlowHTTPTest, Hulk, GoldenEye), Heartbleed, web attacks (Damn Vulnerable Web App, XSS, brute-force), infiltration of the network from inside, botnet, and Distributed Denial of Service with port scanning (Low Orbit Ion Canon).

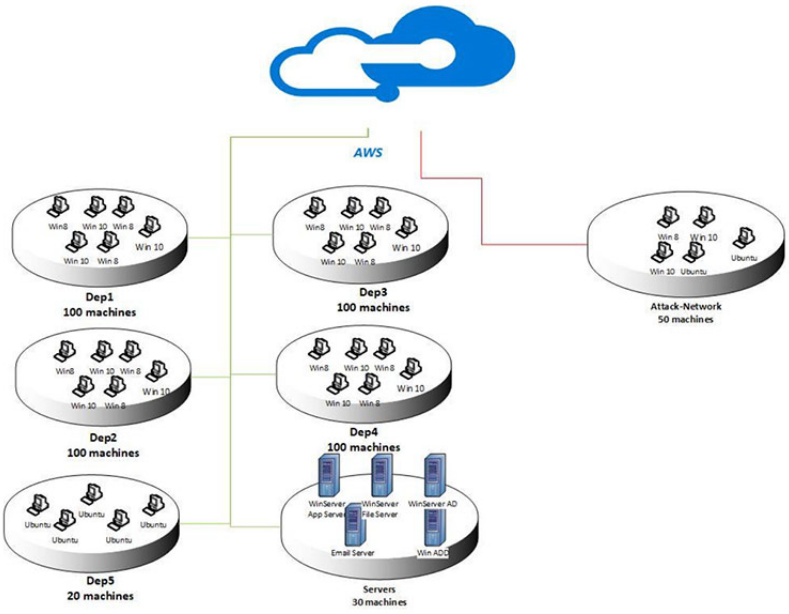

Figure 2.5: Network topology of CSE-CIC-IDS2018.

However, the attacking and the victim networks have a completely different architecture, as shown in Figure 2.5. The attacker uses an infrastructure of 50 machines, while the victim organization has 5 departments and includes 420 machines and 30 servers. The dataset includes the network traffic and system logs of each machine, along with 80 features extracted from the captured traffic using CICFlowMeter-V3 [27]. CSE-CIC-IDS2018 is a more complete dataset than CICIDS2017, with more data and a different network topology.

2.2.3 Performance metrics

Several metrics are used to describe the performance of a classifier. Table 2.5 summarizes the four possible outcomes of a detection.

Table 2.5: Confusion Matrix

		Predicted	
		Normal	Attacks
Actual	Normal	True Negative	False Positive
	Attacks	False Negative	True Positive

- Detection rate (or "true positive rate", "recall", "sensitivity") is the proportion of attacks that are correctly detected.

$$Detection\,rate = \frac{TP}{TP + FN} \tag{2.1}$$

- False positive rate (or "false alarm rate") is the proportion of normal traffic incorrectly flagged as attack.

$$False\,positive\,rate = \frac{FP}{TN + FP} \tag{2.2}$$

- Accuracy is the fraction of correctly identified results (attack and normal traffic). In multiclass classification, accuracy is equal to the Jaccard index, which is the size of the intersection divided by the size the union of the label sets.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.3}$$

- Precision (also called positive predictive value) is the proportion of identified attacks that are indeed attacks.

$$Precision = \frac{TP}{TP + FP} \tag{2.4}$$

- F1-score is the harmonic mean of precision and recall (previously called "detection rate").

$$F1\text{-}score = \frac{precision \times recall}{precision + recall} \tag{2.5}$$

- The ROC curve is the plot of the true positive rate against the false positive rate (see Fig. 2.6). The area under the ROC curve is the probability that a randomly chosen positive is ranked before a randomly chosen negative.

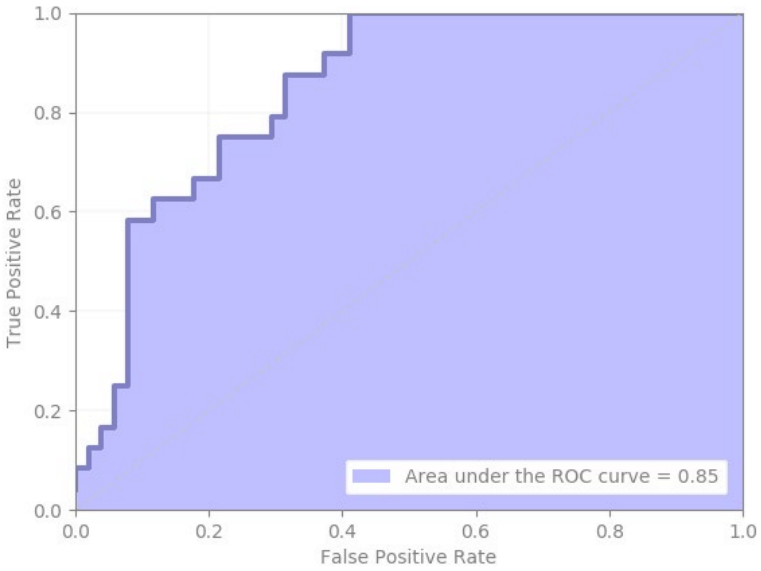

Figure 2.6: ROC curve example.

Chapter 3

State of the art

3.1 Multilayer Perceptron

A Multilayer Perceptron (MLP) is a feedforward artificial neural network, composed of an input layer, one or several hidden layers, and an output layer (Fig. 3.1). Each layer is itself composed of one or more neurons, whose role is to apply a function called "activation function" (generally a *sigmoid*, or a *tanh* function, or more recently a rectifier linear unit) to its input. A value called "bias" is also added by the neuron, and each connection between two nodes possesses a weight in relation to its importance in the network. When a MLP has more than one hidden layer, it is considered as a deep MLP.

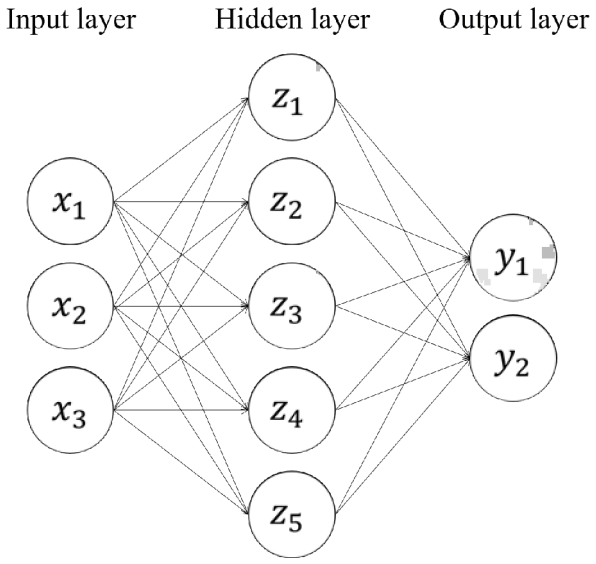

Figure 3.1: Multilayer perceptron with one hidden layer.

Training a MLP means optimizing the values of weights and biases in order to minimize the result of a loss function (like the mean squared error). Gradient descent is

a popular algorithm to train a MLP. This technique calculates an error function by comparing the output values of the network to the already known correct answer and is thus supervised learning. It then updates the weight of each connection and the bias of each hidden neuron according to the amount that they contributed to the error. This part of the training is famously known as the "backpropagation algorithm". The value of the loss function converges to a global (or local) minimum after a certain number of iterations. When this is the case, the output values of the network are very close to the correct answers, and the MLP can be considered as trained. However, if this process is repeated too many times, there is a risk of overfitting – when a model is too closely fit to a dataset. The trained neural network is then too rigid and becomes useless when applied to a real problem with data never seen before.

Stochastic gradient descent is a well-known variation of this technique, where the error function is only computed for a subset of the entire dataset [28]. This approximation of the error function leads to a faster convergence, hence the popularity of this method. It is also particularly suitable for deep neural networks, which require larger datasets than shallow networks. However, other optimization algorithms have been proposed recently like Adagrad (2011) [29], Adadelta (2012) [30], Adam (2015) [31], etc. These techniques are more flexible, with adaptive learning rates, and show impressive results when applied to deep neural networks. These deep neural networks were traditionally pre-trained with a greedy layer-wise training, as introduced by [32]. The goal was to initialize weights more efficiently than random values, to attain a faster convergence. Nonetheless, nowadays, pre-training is only performed for datasets where most of the data is unlabeled. Moreover, the activation function "Rectifier Linear Unit" ($ReLU$) [33], in addition to dropout, a regularization technique that randomly deletes neurons and their connections [34], proved to get better results without pre-training in other scenarios.

MLPs were widely used for pattern detection and handwriting recognition in the 1980s, before being replaced by other machine learning algorithms such as support vector machines in the 1990s. Like every type of neural networks, they were partially abandoned during this period, before regaining popularity with the rise of deep learning (an alias for neural networks with several hidden layers) that began in 2006. Being one of the easiest type of neural networks to implement, MLPs are commonly used for intrusion detection.

Palenzuela et al. (2016) [35] show how they designed their neural network by choosing a configuration among several options. They used a binary classifier: their goal is not to classify the attack (DoS, Probe, R2L, U2R), but to determine if a packet corresponds to a malicious activity or a benign one. Their preprocessing stage reduces the number of parameters in the KDD Cup 99 dataset from 41 to 38. They then use this dataset with seven different MLP configurations: from zero to three hidden layers, with different numbers of neurons. This comparison can be used to measure the impact of each topology on the results. Finally, they compare the accuracy of every configuration to conclude that the MLP with a single hidden layer of 10 neurons gives them the best results: a 39-10-2 structure (each number represents here the number of neurons per

layer), yielding a 99.85% accuracy and 0.17% false positive rate. The authors used part of the training dataset here to evaluate their performance instead of using the test set provided for this purpose. To improve even further the accuracy of this topology, the authors made a strong trade-off by adding a bias of 0.9999 to the output for attacks. This trick increases the accuracy to 99.99%, but also the false positive rate to 9.83%.

Mowla et al. (2017) [36] designed an IDS for Medical Cyber Physical Systems. Their classifier can be easily transposed to a regular network since it is trained and tested with KDD Cup 99. The neural network is a deep MLP for binary classification with two hidden layers. The idea of transforming a 5-class problem into a combination of 2-class problems is particularly interesting. The neural network starts to classify the monitored data as an attack or not. If this is an attack, it tries to classify it as a DoS attack or another undetermined attack. If the attack is undetermined, it tries to classify it as a probe attack or another unknown kind of attack. Finally, this process is repeated one last time to classify the data as a R2L attack or a U2R attack. These different kinds of traffic seem to be sorted in order of probability, with the most probable traffic tested first. Note that only 1200 attacks are used during the evaluation process along normal traffic. Even among these attacks, only 16 subtypes are represented among the 38 included in the KDD Cup 99 dataset. Mowla et al. also explain the choice of this "evolved 2-class" approach by comparing the training time and the number of correctly classified instances of their solution with a regular 5-class classifier. The "evolved 2-class" approach appears to more efficient, with a training time of 12.14 seconds and a detection rate of about 98.7%, compared to the 264.19 seconds and 98.0% of the 5-class classifier (hardware used to perform these computations was not specified).

Potluri and Diedrich (2016) [37] propose a deep MLP architecture, with a 41-20-10-5-5 structure. They start with a usual two-step preprocessing stage of the NSL-KDD dataset: first, a conversion of non-numerical features, and then their normalization between 0 and 1. The 41 features of the NSL-KDD dataset are sent into the input layer of the deep MLP. The authors use a greedy layer-wise pre-training stage with autoencoders (see next section), as described by [38]. The goal of this stage is to initialize the weights of the network in a more efficient way than random values. While this technique was widely popular in the early stages of deep learning, it is now very often outperformed by recent initializers such as the He initializer [39]. The first hidden layer is pre-trained as an autoencoder, which selects 20 features out of the 41 of the input layer. The second hidden layer, which is also pre-trained as an autoencoder, selects then 10 features out of the previous 20 ones. Finally, the third hidden layer selects 5 features out of the previous 10 ones and applies a softmax function for a proper classification. This layer is the only one that is pre-trained in a supervised way (DoS, Probe, R2L, U2R or normal). After this pre-training, the whole network is trained by performing backpropagation – and apparently using a stochastic gradient descent. The accuracy of the neural network is evaluated on the NSL-KDD test set where it obtains 97.5% with 2 classes. Finally, the authors evaluate the computing performance of their network, by comparing its training time on different hardware (CPU in serial or parallel mode, GPU, CPU+GPU). Surprisingly, the best training time is achieved by a CPU (Intel Core i7 4790 in parallel

mode) with 108.85 seconds. This could be explained by the fact that the GPU (132.51 seconds) used in this experiment is a mid-end laptop GPU (Nvidia GeForce GTX 960M) not fitted for deep learning. Interestingly, combining a CPU (Intel Core i7 4270 HQ) and the previous GPU does not improve performance (145.59 seconds) over the GPU alone. One CPU core is dedicated to data transfer and the other three are dedicated to intrusion detection. The GPU probably finishes its share faster than the other three cores, hence the extra training time.

Kim et al. (2017) [40] introduce another deep neural network solution. They first preprocess the entries of their dataset (KDD Cup 99 in this case) with conversion and normalization like the previous IDS. Their architecture has 4 hidden layers and a total of 100 hidden neurons, built and tested using TensorFlow. The authors chose the Rectified Linear Unit ($ReLU$) function as the activation function of the hidden layers. This choice greatly improves the learning speed of the network, $ReLUs$ being several times faster than *tanh* or *sigmoid* functions according to [41]. They also added a more recent optimization method, the Adam optimizer (2015), which aims to combine the advantages of Adagrad and RMSProp methods [31]. Evaluation is conducted for several training sets with different proportions of attacks (from 10% to 90%). The test set remains the same (the entirety of KDD Cup 99), and the results are very similar for each training set. On average, they obtain a 99% accuracy, a 99% detection rate with a 0.08% false alarm rate. The authors used an Intel i5 3.2 GHz CPU, 16 GB of RAM, and an Nvidia GeForce GTX 1070 GPU.

3.2 Autoencoder

An autoencoder (Fig. 3.2) is an artificial neural network, which learns an encoded (compressed) representation of the input data. These networks have a distinctive hourglass shape, with a first layer that has the same size as the last layer, but fewer neurons in their hidden layers.

The goal of a basic autoencoder is to reconstruct the input data, so that x'_i is as close as possible to x_i. In other words, an autoencoder learns an approximation of the identity function. The trick is that the limited number of hidden neurons forces the network to find patterns, structures in the input data to be able to encode it (hidden layer), and then to decode it (output layer). The backpropagation and gradient descent techniques are the standard methods to train an autoencoder. After measuring the deviation of x'_i from x_i, the error is backpropagated to update the weights of the network. But contrary to MLPs, an autoencoder does not need labeled input data: it uses the input as a model for correcting its output. This training process is thus unsupervised.

Besides, hidden neurons in the middle layer of regular autoencoders tend to be activated too frequently during the training process. A variation called “sparse autoencoder” introduces a sparsity parameter to lower the activation rate of these neurons [42]. This sparsity has the same effect than limiting the number of neurons in the hidden layers: it is a constraint to discover interesting structures in the input data. In a sparse autoencoder, the number of hidden neurons can be even greater than the number of neurons

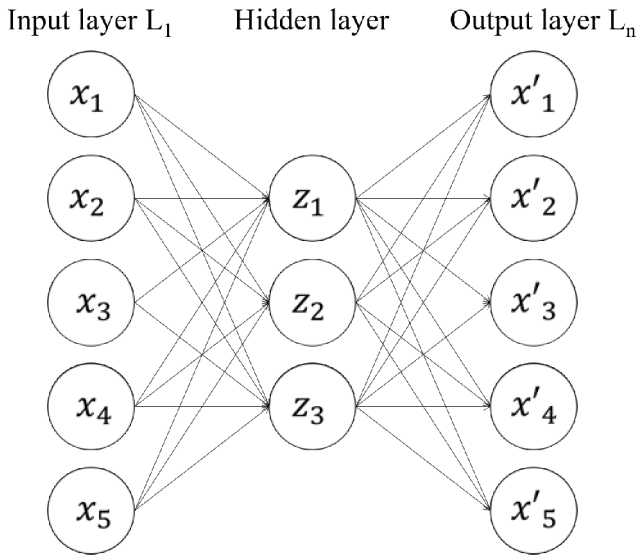

Figure 3.2: Autoencoder with one hidden layer - $card(L_1) = card(L_2)$.

in the input layer. A network with a high level of sparsity is capable of capturing more local features, whereas a network with a low level of sparsity is more general.

Autoencoders are particularly useful for non-linear dimensionality reduction and some deep neural networks pre-training. However, they can also be implemented as independent classifiers for intrusion detection.

Niyaz et al. (2016) [43] present an interesting solution with the use of self-taught learning. The idea of self-taught learning, as described in [44], is to learn patterns from unlabeled data to use them for the supervised learning task. The authors chose a sparse autoencoder for unsupervised feature learning among other solutions due to its ease of implementation and overall good performance. The sparse autoencoder tries to reconstruct its input using the backpropagation algorithm and the *sigmoid* function as the activation function. They then use this feature representation for classification with a softmax regression. A performance comparison is shown, where the 2-class classification results outperform these of the 5-class model in every metric (88.39% accuracy for 2-class 79.10% accuracy for 5-class on NSL-KDD). According to the authors, performance can be enhanced with a stacked autoencoder for unsupervised learning, and NB-Tree, Random Tree, or J48 for classification. According to our own research, this is particularly true for rarer classes such as U2R where more deterministic algorithms perform better than neural networks.

Yousefi-Azar et al. (2017) [45] describe a deep autoencoder solution made with Weka, able to manage intrusion detection as well as malware classification. The goal of the pre-training phase is to obtain the initial weights for the fine-tuning stage. The authors use restricted Boltzmann machines, stacking them on top of each other to reproduce the topology of their deep network: a 150-90-50-10-50-90-150 structure. Every

neuron uses *sigmoid* as the activation function, with the exception of the ten nodes in the middle of the network that are linear. The whole network is then fine-tuned with backpropagation, using the cross-entropy error as the loss function. Finally, NSL-KDD data are preprocessed with the replacement of symbolic features with one-hot encoding, and Principal Component Analysis (PCA). PCA is a popular statistical method to reduce the dimensionality of a dataset. It identifies important uncorrelated variables (principal components) and helps to remove the ones that only have a minor influence on the outcome. The authors obtain a 83.34% accuracy on NSL-KDD with this method.

3.3 Deep Belief Network

A Deep Belief Network (DBN) is an artificial neural network composed of multiple layers of hidden units (Fig. 3.3). The connections are directed from the top layers to the lower layers, with the exception of the two top layers that are interconnected (note the direction of the arrows in Fig. 3.3). This latter feature is the most visual difference between a DBN and a deep MLP. The *sigmoid* function is a common activation function for these networks.

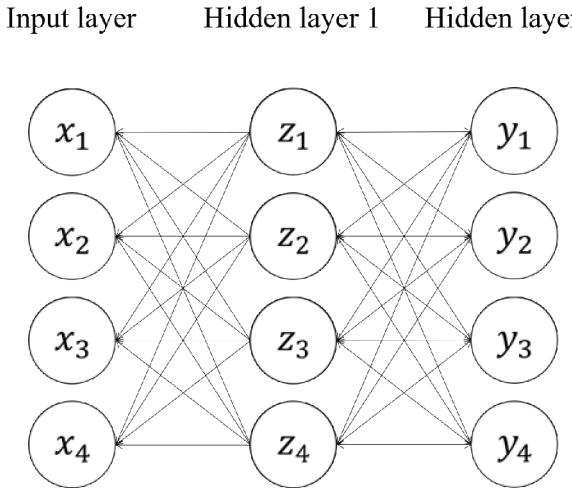

Figure 3.3: Deep Belief Network with two hidden layers.

DBNs were the first type of deep neural network, and appeared in [32]. The proposed learning method consisted of a stage of pre-training to initialize the weights of the network (instead of a random initialization), and another stage of fine-tuning. They used a greedy algorithm to pre-train DBNs one layer at a time, considering each layer as a Restricted Boltzmann Machine (RBM) and only taking into account the lower layers. In other words, a trained layer is used as an input layer for the next untrained RBM, until it reaches the top of the DBN. Other regular learning techniques are then used to fine-tune the network, such as gradient descent.

[38] demonstrated that autoencoders could also be used as the building block of a

DBN, instead of RBMs, with the same learning process. Famous between 2006 and the early 2010's, DBNs have since lost popularity [46]. This is probably due to the efficiency of $ReLUs$ and dropout, to tackle the vanishing gradient problem, which was the main advantage of DBNs [47].

Alrawashdeh and Purdy (2016) [48] describe an attempt toward a real-time classifier with a RBM-based DBN. The authors start with the usual pre-processing stage, where non-numerical data of KDD Cup 99 are converted and, in this case, are expanded from 41 to 122 features. This expansion is due to the one hot encoding, a conversion method we will discuss in section IV. The numerical features are then normalized between 0 and 1. A DBN is trained with a greedy layer-wise training, using Gibbs sampling and contrastive divergence to reduce the number of features. The authors test several topologies: a RBM alone, a DBN, and a DBN with logistic regression (a softmax layer for classification). They use a laptop with 4 GB of RAM and a 2.1 GHz CPU running Visual Studio 2013. The best results are obtained by the latter topology, with a 122-72-52-40-5 structure (97.9% accuracy, 0.51% false positive rate). However, this classifier is too slow to be used in real-time detection. The authors propose a simplified DBN, with a 122-72-5 structure, combined with a training of only 10 epochs (the number of full training cycles on the training set) to reduce the testing time (0.70 second for each batch of 1000 records).

Liu and Zhang (2016) [49] combine RBM-based DBN with extreme learning machine. The authors preprocess data from the NSL-KDD dataset with the conversion of non-numerical entries. The 41 initial features are expanded into 122 attributes (one hot encoding [50]), which are normalized between 0 and 1. The DBN has four hidden layers of respectively 110, 90, 50 and 25 units. It is trained in a greedy manner, one layer at a time, with Gibbs sampling and contrastive divergence. The neural network was developed with the Deep Neural Network module [51] and the Deep Learning Toolbox [52] with Matlab 2012b, running on a 3.30 GHz Intel CPU and 16 GB of RAM. Once the weights are initialized, the DBN is fine-tuned with the use of extreme learning machine. Extreme learning machines are single hidden-layer feedforward neural networks that makes the training process equivalent to solving a linear system. According to the authors, this technique halves training time (22.7s) compared to a DBN without extreme learning machine (47.2s), though the detection rate decreases from 92.4% to 91.8%.

3.4 Recurrent Neural Network

A Recurrent Neural Network (RNN) is an artificial neural network whose nodes send feedback signals to each other (Fig. 3.4). This term covers numerous architectures using this idea: hierarchical RNN, continuous-time RNN, bi-directional RNN, the popular Long Short-Term Memory (LSTM) network, etc. RNNs possess an internal memory that allows them to have a natural notion of order in time.

These networks cannot be trained with a regular backpropagation technique, which requires the connections between the neurons to be feedforward only. The most common method is a generalization of this technique, called "backpropagation through time".

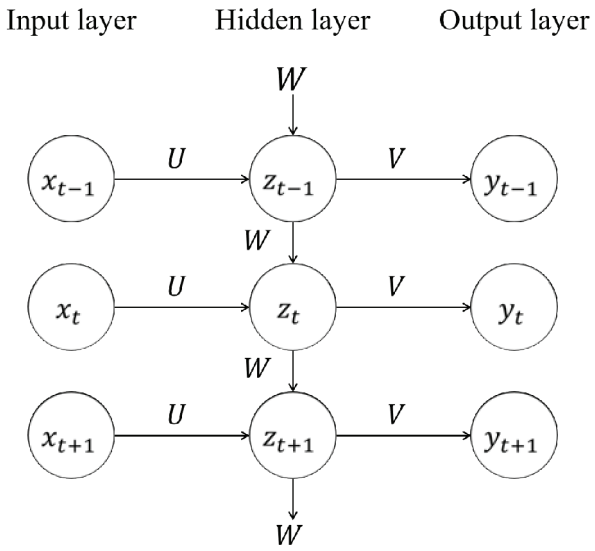

Figure 3.4: Basic Recurrent Neural Network with one hidden layer.

However, this technique has difficulties distinguishing local from global optimas – much more than feedforward networks. It can be strengthened with global optimization methods, and particularly genetic algorithms. The *sigmoid* and *tanh* functions are two popular activation functions for LSTMs and RNNs.

While the ability of RNNs to have information persistence through the network can be appealing, in practice, they have difficulties with long-term dependencies [53]. LSTM networks solve this problem with a particular repeating module containing four interacting layers instead of one. The idea is to keep the information as unmodified as possible through the entire network, with carefully managed interactions. Therefore, older dependencies are rather undamaged by new information.

RNNs, and mostly LSTM networks, are often used to deal with sequential problems, such as translation, speech recognition, or even text generation. LSTM is the most popular type of RNN for intrusion detection.

Staudemeyer (2015) [54] studies how to apply LSTM to intrusion detection as a classifier. The author experiments with different parameters at each step of the configuration, starting with the LTSM structure itself. Four LSTM topologies are benchmarked to finally keep a compromise between computational cost and detection performance (four memory blocks containing two cells each, with an input layer fully connected to the hidden and the output layers). Likewise, different learning rates and various numbers of epochs are tested and compared based on the area under the curve of the ROC. A LSTM classifier tuned with the best parameters is then evaluated in several environments – focusing for example on one category of attacks. The last experiment consists of performing an analysis with a minimal sets of features (4 or 8 features only). The minimal set of 4 features contains the most important features of KDD Cup 99 for this classifier: *service*,

src_bytes, *ds_host_diff_srv_rate*, and *dst_host_rerror_rate*. The minimal set of 8 features adds *dst_bytes*, *hot*, *num_failed_logins*, and *dst_host_srv_count* to this list. The LSTM classifier obtains nonetheless a very similar accuracy (93.72% with only 4 features) compared with the full 41 features KDD Cup 99 dataset (93.82%).

Kim et al. (2016) [55] introduce another IDS using a LSTM. They set the values of time step size, batch size and number of epochs (100, 50, and 500 respectively) to find optimized values for two hyperparameters: learning rate and number of hidden neurons. The authors increment the learning rate between 0.0001 and 0.1 to set it at 0.01 (a commonly accepted value). They then progressively change the number of neurons in the hidden layer from 10 to 90 to finally keep the value of 80 neurons. Stochastic gradient descent is used to train the network with mean square error as the loss function. This training runs on an Nvidia GeForce GTX Titan X, an Intel Core i7-4790 3.60 GHz, with 8 GB of RAM on Ubuntu 14.04. In 2017, a similar team with Le et al. [56] continues the previous work on LSTM classifiers, focusing on gradient descent optimization. They compare six different solutions (Adagrad, Adadelta, RMSProp, Adam, Adamax, Nadam) using KDD Cup 99 for the experimental results of their IDSs. Their conclusion shows that Nadam is not only the best overall optimizer (97.54% accuracy, 98.95% detection rate), but also the best one to classify each type of attack individually. However, with a very high false positive rate of 9.98%, this classifier is not suitable for a real-world deployment.

3.5 Self-Organizing Maps

A Self-Organizing Map (SOM), also known as Kohonen network, is an artificial neural network trained with competitive learning – unlike the previous neural networks and their error-correction learning. Competitive learning means that neurons compete with each other to represent the input as accurately as possible. SOMs have a simple architecture, with an input vector connected to a grid of output neurons (Fig. 3.5). Each neuron is associated with a position in the map space and a weight vector of the same dimension as the input vector. SOMs do not possess activation functions like the previous types of networks.

After an input vector is selected, the algorithm computes its weighted distance with every neuron (usually using Euclidean distance). The neuron whose weight vector is the closest to the input vector becomes the best matching unit. This weight vector and the weight vectors of its neighbors are then updated to reduce their Euclidian distance with the input vector. This distance is defined by a learning rate, which decreases with each iteration. Likewise, the radius of neighbors around the best matching unit also decreases after each iteration. These steps are repeated for each input vector for a predetermined number of iterations. The SOM thus converges gradually towards its trained form. This learning process is unsupervised, but there are supervised variants of SOMs with an output layer. During the test process (or mapping in this case), the node with the closest weight vector determines the class of the input vector.

Unfortunately, SOMs have two downsides: their architecture is static, and they lack

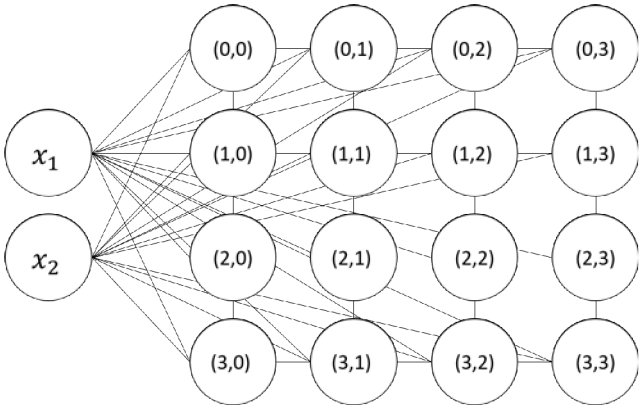

Figure 3.5: Self-Organizing Map with a two dimensional input vector and a 4x4 nodes network.

capabilities to represent hierarchical relations [57]. These limitations are addressed with the Growing Hierarchical Self-Organizing Map (GHSOM), by [58]. The idea of GHSOM is dual: on the one hand, to create sub-maps hierarchically contained in other maps to represent sub-parts of the data. On the other hand, the size of every map and sub-map is dynamically adapted to the dimension of the input data.

SOMs' main feature is the ability to represent high-dimensionality data in just one or two dimensions. It is particularly popular in cybersecurity, compared to other areas. SOMs are indeed one of the most common types of neural networks for IDSs.

Kiziloren and Germen (2009) [59] experimented the effects of PCA applied to SOMs. The authors convert non-numerical parameters of KDD Cup 99 and randomly extract 1000 vectors. They then apply PCA to these vectors, and experiment with different sizes of principal component vectors, varying between 2 and 15. The average detection rate of the system increases for sizes from 2 to 10, and then decreases when the size is higher than 11. The optimal value of 10 is selected. By reducing the dimensionality of the input data, PCA does not only speed up the classification process, but also, coupled with SOM, obtains a better success rate than SOM alone: 98.83% detection rate with PCA (size = 10), and 97.76% without PCA. This is an important result regarding simple SOMs, which will be generalized in further classifiers.

McElwee and Cannady (2016) [60] also employ PCA with SOM, and add another layer of preprocessing with binary filtering. This technique, originally introduced for fraud protection [61], aims to increase the true positive rate at the expense of the false positive rate. Unlike other preprocessing steps we have seen before, they convert continuous data of KDD Cup 99 into symbolic data (discretization). They then delete duplicated entries created by this discretization step. The authors argue that categorization of these data is more important than knowing their exact values. Two binary filtering models are evaluated. The first one is a combination of three neural networks using 1) preprocessed data only; 2) PCA feature extraction applied to preprocessed data; and 3) Independent Component Analysis (ICA), an extension of PCA, applied to pre-

processed data (see section V for more details). The other model is a combination of four neural networks without feature extraction. Surprisingly enough, the latter model outperformed the first one. Normal records are supposed to be eliminated by this binary filtering process, which could be considered as a first classifier. The outcome is then sent to a SOM that categorizes the records into 26 clusters. Experiments show that the two layers of preprocessing (discretization and deduplication, and binary filtering) highly reduce the number of records (an overall reduction of 97.6% and 99.5% of the inputs respectively). The reduction of inputs also leads to a faster build time for the SOM: this time goes from 194 minutes without binary filtering and PCA to 16 minutes with these two techniques (hardware was not specified). The quality of clustering is also improved, with a direct impact on the overall quality of the SOM according to the authors. While the SOM is the real classifier of this paper, we will use the metrics given for binary filtering in the summary in section V. This whole study was conducted on Weka 3.7.13 Java API.

Cheng and Wen (2010) [62] designed an hybrid and real-time IDS with PCA and a SOM. Unlike the two previous papers, PCA is not used here primarily to reduce the dimensionality of the dataset, or to improve the amount of data treated per second. The authors employ PCA as an anomaly detection algorithm, capable of multi-class classification. However, anomaly detection, and PCA in particular, produces a high false positive rate. The goal of the hybrid approach is to reduce the false positive rate by adding a misuse detection algorithm: a SOM in this case. This particular topology (anomaly then misuse detection) is selected considering the IDS performance. Indeed, the SOM only re-checks connections tagged as "attack" by the anomaly detection algorithm. The authors only use the 34 numerical features of the KDD Cup 99 dataset (no symbolic feature). They randomly select 7000 connections for training, and 90277 for testing. Several square of Euclidean distance values are tested for PCA, and the detected attacks are sent to the SOM (a 5*5 map, trained on 2000 epochs with a learning rate of 0.1). In the end, they obtain approximately a detection rate of 98.70% while lowering the false positive rate with the misuse detection step. The test set is processed in 236.7 seconds with anomaly detection and in 274.9 seconds with misuse detection on an Athlon 3000+ CPU with 1.5 GB of RAM.

Ippoliti and Zhou (2010) [63] describe an improved version of GHSOM, called A-GHSOM, for intrusion detection. GHSOMs solve several recurrent problems with SOMs, such as expensive computation and static architecture. This enhanced version aims to fix weaknesses with the GHSOM model itself by adding four new features. First, i) the training process uses a threshold error value instead of Euclidean distance between vectors or mean quantization error. According to the authors, this threshold allow them to capture accurately anomalies that would have vanished with mean quantization error for instance. The input data is also ii) dynamically normalized between 0 and 1, with a continuous monitoring to detect new minimum or maximum values. A feedback-based quantization error threshold adaptation iii) adjusts threshold error values for each existing node and adds new nodes if needed. Neurons are also iv) monitored in order to attribute them a confidence rating. Traffic of neurons with low confidence is rejected by

the rest of the map. This classifier is trained and tested with KDD Cup 99. During the training process, the GHSOM dynamically grows to fit the training dataset. This means that this architecture can adapt to changes in the input data over time. However, the authors indicate that this model is not suitable for online learning with live data (no test time was specified). The A-GHSOM achieves a 99.63% accuracy (94.04% accuracy on unknown attacks) with a 1.8% false positive rate.

Salem and Buehler (2013) [64] designed their own enhanced GHSOM, knowing the previous work of Ippoliti and Zhou. They also introduce four improvements to the GHSOM model. The first one is a meaningful initialization (not a random one) of the weight vectors on a new map, based on minimum and maximum boundaries of the input dataset. They also add a new heterogeneity threshold to provide a more stable growth and robust hierarchical topology. The third feature is the fusion of similar best matching units, whether they are weak or coherent, in order to reduce their number and speed up classification. Finally, they introduce a confidence threshold to detect new or unknown connections. The authors carry out several evaluations to compare their model and specific features to other GHSOMs. Their enhanced GHSOM achieves better results on NSL-KDD than the A-GHSOM on KDD Cup 99, with 99.9% accuracy, 99.9% detection rate, and 0.123% false positive rate. Training and testing are also conducted not only with NSL-KDD, but also interestingly with the real-time dataset SecMonet (data acquired from live university traffic). However, no test time is specified in this paper either.

3.6 Radial Basis Function Network

A Radial Basis Function Network (RBFN) is an artificial neural network that uses radial basis functions as the activation function. It is composed of an input layer, one single hidden layer (contrary to MLPs), and an output layer (Fig. 3.6). Radial basis functions are a popular way to approximate functions with multiple variables by a combination of simpler functions. Their output depends on the distance between the input and a stored vector. A RBFN measures the similarity between the input data and "prototypes" (stored examples from a dataset) to perform classification.

Each hidden neuron computes the Euclidean distance between the input and its stored prototype. The output of this neuron is 1 if they are equal, and falls off rapidly towards 0 otherwise. These hidden neurons typically use a Gaussian function, with a specific beta coefficient to adjust the width of the curve. Each output neuron then computes a weighted sum of the output of the hidden nodes. In the end, input data are classified in the category of the output neuron with the highest score.

In a RBFN, the training process covers the selection of the prototypes, beta coefficients for each hidden neuron, and weights for each connection between hidden and output neurons. The selection can be random or use clustering algorithms like k-means. Usually, the k-means algorithm finds the right beta coefficient. The weight values are then fine-tuned with a variation of backpropagation dedicated to RBFNs. RBFNs are mostly used in combination with other techniques like support vector machines.

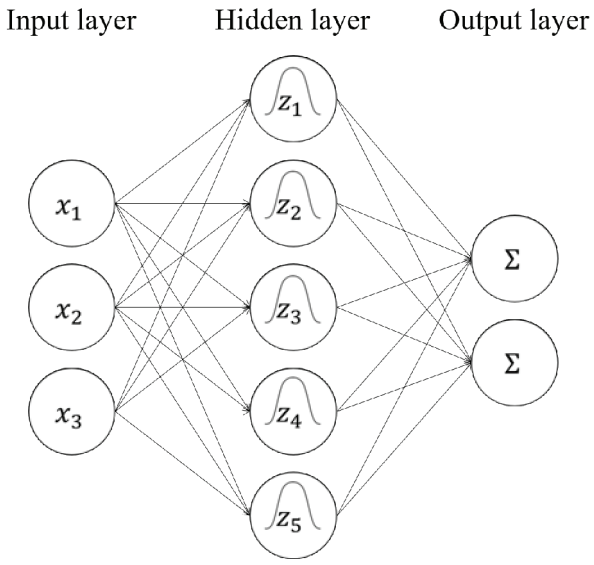

Figure 3.6: Radial Basis Function Network.

Yichun et al. (2012) [65] combine RBFN with an Artificial Immune System (AIS) optimization. The term "AIS" aggregates rule-based machine learning systems inspired by the principles of immune systems. This refers to the function of the body that distinguishes self from non-self elements. More specifically, the authors use an artificial immune recognition algorithm based on clonal selection to simulate the learning process of a biological immune system with antigens and antibodies. This process leads to optimize values for prototypes of the RBFN's hidden neurons. They then adjust weights between hidden layer nodes and output nodes with the recursive least squares method, because of its proper global convergences. This IDS is trained and tested with KDD Cup 99 with a special process to produce correct antibodies. Experiments were conducted both in real-time and offline, although only offline test results are shown (83.7% detection rate). Hardware and test time were not specified.

3.7 Adaptive Resonance Theory

Adaptive Resonance Theory (ART) is a family of artificial neural networks with the capability of learning new information without necessarily forgetting things learned in the past [66]. ART encompasses a wide variety of neural networks, including ART 1, ART 2, ART 3, ARTMAP, Fuzzy ART and many others.

ART networks generally use unsupervised learning, except for the algorithms with the suffix "MAP" (supervised learning). ART 1 is the simplest ART architecture, which is why it will be described here. The more advanced versions of ART have a complex architecture; the reader is referred to [67] and [68]. ART 1 is composed of a short-term memory layer (F1), a recognition layer (F2) that contains the long-term memory, and a

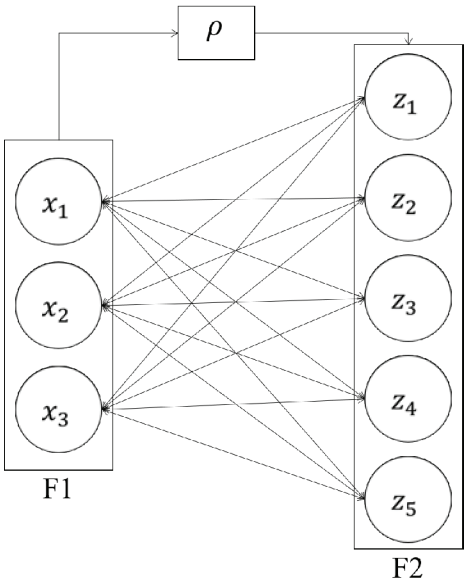

Figure 3.7: Adaptive Resonance Theory.

vigilance parameter (ρ with $0 < \rho \leq 1$) that controls the level of detail of the memories (Fig. 3.7). Additionally, there are two sets of weighted connections: one from each unit of F1 to all units of F2 (bottom-up weights), and another one from each unit of F2 to all units of F1 (top-down weights).

During the training process, the top-down weights are first initialized to 1, and the bottom-up weights are initialized to $1/n + 1$ (where n is the number of nodes in F1). Then, each node of F2 receives a linear combination of the input vector from F1 with their bottom-up weights. The node with the highest linear combination then calculates the linear combination of the input vector with its top-down weights. If the division of this second linear combination with the input vector is greater than ρ, then the input is associated with this node. Both bottom-up and top-down weights are updated. Otherwise, this process is repeated with the second node with the highest linear combination, until there are no more output nodes to test. In this case, a new node is created especially for this input vector. This mechanism allows ART networks to learn entirely new data without degrading older memories. Once the network is trained, it classifies data by comparing the similarity of the vector input with its own output nodes.

ART 1 is not used anymore, but two other subtypes of ART are still popular for intrusion detection. ART 2 is a more complex version of ART 1, with a refined F1 layer, that gives better results and supports continuous inputs. The ART 2-A variation, with faster training and testing time, is often preferred to its counterpart [67]. Fuzzy ART is another version of ART that implements fuzzy logic for better generalization [68]. There are only two types of ART currently used by IDSs: ART 2-A and FuzzyART.

Han (2019) [69] employs a variant of ART 2-A to deal with the categorical data of KDD Cup 99. Here, data is simply normalized but without the usual conversion of non-numerical features. The improved ART 2-A consists of three layers: F0, F1, and F2. F0 receives the input data, F1 acts as a comparison layer, normalizing numerical attributes, and F2 contains the long-term memory weight vectors. The difference between this version and the original ART-2A is the concept of mode, which allows the management of similarity between categorical values. After the training process, output neurons in the F2 layer are labelled depending on the most frequent type of data they contain. This ability is promising to detect and classify correctly completely unknown categories of attacks. For instance, an output neuron with a majority of DoS attacks would be labelled as "DoS". The author tries 10 different values of vigilance, combined with a classic learning rate of 0.1. The best results are achieved with a vigilance value of 0.9991. The ART-2A classifier is trained with these parameters during 10 epochs and obtains a detection rate of 99.57%.

Ngamwitthayanon and Wattanapongsakorn (2011) [70] exploit the capabilities of Fuzzy ART for their binary classifier. This technique combines fuzzy logic with ART and is used, in this context, as a clustering algorithm. The authors make three assumptions about where normal data should be found after clustering: 1) they belong to a cluster in the data; 2) they lie close to their closest cluster centroid, whereas anomalies can be found far away; and 3) their clusters are large and dense, unlike anomalies. The main hypothesis is that all data that do not respect these three assumptions are considered to be anomalies. The training process of this Fuzzy ART classifier is close to a regular ART training. A final step is added to this process: one shot fast learning. This technique's idea is to nullify choice parameter and maximize learning rate to 1, so that every input instance is presented to the classifier only once. The number of clusters is also restricted to 1 (only normal data), which reflects the first assumption. The authors perform three experiments, with different attack rates: 1%, 10%, and 20%. These experiments are conducted on a 2 GHz Intel Core 2 CPU with 2 GB of RAM on Matlab R2009b. On average, the Fuzzy ART classifier obtains a detection rate of 99.16%, 99.05%, and 99.06% respectively, with a false positive rate slightly above 1% (1.25%, 1.55%, and 1.46%).

3.8 Comparison of different intrusion detection systems

Neural networks are a popular choice among other machine learning techniques – with support vector machines and clustering algorithms. Nowadays, it is the most employed technique for anomaly-based IDSs. It must be emphasized that computer vision, speech recognition, and natural language processing are by far the main research topic in neural networks. A very successful type of neural networks in image and video recognition is the Convolutional Neural Network (CNN). However, if CNN is a widespread solution for this topic, it is not the case for intrusion detection. Indeed, CNNs are very rare in this field and do not yield the same performance with IDSs than with image recognition. This is the case with other types of neural networks (Neural Turing Machine, Deep Residual Network, Echo State Network, etc.) that are, in the intrusion detection field, either

completely absent or underrepresented. The most popular types for IDSs are SOMs and MLPs. It is harder to differentiate between the others: RBFs are uncommon, RNNs and ARTs are rarer, but not as much as DBNs. Autoencoders were often used in deep neural networks for pre-training purpose, but hardly on their own.

A word of warning must be added to the results present in Table 3. Detection rate, accuracy and false positive rate have been carefully compiled, sometimes with personal figures calculated with the statistics given in the article. However, it is legitimate to be doubtful of some results since they cannot be verified. In some instances, it is difficult to precisely identify figures written in the articles: they could either be detection rate or accuracy. Moreover, authors do not carry out tests in the same way: they can re-use another part of the training data, instead of exploiting the test data planned by their datasets. Sometimes, they use a specific percentage of attacks in the test traffic, or ignore certain subtypes of attacks. The disparity between these datasets is also problematic. As pointed out in its description, KDD Cup 99 has several flaws that have been widely documented over the past years. In most cases, authors try to mitigate the errors of the dataset (for example, by duplicating data in the preprocessing phase). However, this mainly leads to the creation of a new personal dataset for each solution. These problems prevent a rigorous comparison of the detection rate, accuracy and false positive rate between the different IDSs, but provide a general idea about their performance. The results obtained on these datasets, though, are unrelated to what these models could obtain on real data. The comparison of these architectures is therefore only valid for KDD Cup 99 and/or NSL-KDD.

High detection rate or accuracy values prove the viability of these solutions in their ability to find malicious data. Unfortunately, the main criticism about anomaly-based IDSs is not poor accuracy but high false positive rate. It is impossible for a system administrator to use an IDS whose several percent of the alerts are false alarms: loss of time would simply be too massive. Nonetheless, certain authors do not value the significance of a low false positive rate. This is why they sometimes prefer to generate a large quantity of false alarms to gain a minimal increase in detection rate or accuracy. Our advice and goal here is to minimize as much as possible false positive rate (ideally under one percent) while keeping a high accuracy in a balanced compromise.

3.9 Open issues and challenges

It should be noted that numerous IDSs presented here are not fully suited for a real-world setting. Indeed, in order to achieve the best possible accuracy, a majority of these IDSs is designed to work in an optimal environment that is not a good representation of real-world networks. An actual deployment of one of these IDSs would cause many problems that are rarely discussed, even though they could provide interesting perspectives and modify certain design choices [71].

First, this deployment could either be centralized or distributed, with a particular number of monitored machines per sensor to be defined. While a centralized solution can benefit from the full power of a computer, distributed instances often run on system

on a chip, which are a lot more computationally constrained. If the probes monitor the same network, one probe could train a new neural network model and propagate it to the others to continuously adapt to the evolutions of the network. Conversely, a centralized IDS means a single point of failure, whereas a distributed IDS is much more resilient. This architecture choice depends on multiple factors, including the goal of the IDS in a global network security plan, or the topology of the monitored network. A combination of these two solutions could translate into a centralized neural network with a high-level view of the network, capable of detecting threats that would go unnoticed by the probes. For a centralized solution, it is still possible to use the probes by having them preprocess the data they send to the server. This reduces both the load on the network and the amount of work assigned to the central server. However, IDSs presented in this section are, when this information is available, always trained and tested on a single computer.

But even in this case, their analyses are often performed offline, with only a small number capable of real-time detection for high-bandwidth networks. While this is also the case with certain commercial cybersecurity solutions, this outcome is not fully satisfying. This would be an important problem in a real-world setting: if the IDS takes too much time to analyze incoming traffic, malicious data could harm the network before being detected. Or, even worse, the saturated IDS could drop packets containing the attack, making its detection impossible. Beyond a certain threshold, the IDS indeed simply ceases to function [72] – a flooding attack that is particularly cheap and effective.

This is why the security of the IDS itself is extremely important. This issue is hardly or not covered at all in the reviewed papers, but it remains a major concern for any IDS. In a context where attackers expect to be confronted to such a protection, an IDS becomes itself an excellent target to neutralize. This security problem is related to the choice of a centralized or a distributed architecture: a centralized IDS would mean a single point of failure, but protecting probes with little computational power could be difficult without interfering with their monitoring. IDS security can be divided in three parts: Operating System (OS) security, hardware virtualization, and network surveillance [73]. Any IDS runs on an OS, which is a new source of potential attacks. A secured configuration of the OS is required, with the removal of every application that is not used by the IDS for example. Hardening the OS is a second step, with many operations depending on the OS to improve kernel security (grsecurity for the Linux kernel) or to reduce user permissions. Virtualization is a good idea to isolate crucial processes from the rest of the OS, with a tradeoff between hardware virtualization (entire OSs) and containers (like LinuX Containers, OpenVZ). While hardware virtualization gives a better isolation, containers are more computationally efficient. Any IDS, whether it is centralized or distributed, should have its own network for data acquisition, alerts feedbacks, and administration (preferably several VLANs).

The information obtained from the analysis of the state of the art in this chapter is used for the design of our own IDS in the following chapter. The optimization of the parameters of the machine learning model is a particularly important topic, where the quality of detection can be improved. The quality of the pre-processing is another point that can be improved. Corrections to the training dataset can help to better identify

classes, especially the rarer attacks such as R2L and U2R.

Chapter 4

Supervised Intrusion Detection

This chapter introduces an ensemble learning approach for classification in intrusion detection. Its application to the KDD Cup 99 and NSL-KDD datasets consistently increases the classification accuracy compared to previous techniques. Section 1 describes a version using only neural networks in a cascade-structured architecture. Section 2 adds many machine learning models for classification and changes the combination rule of the different models.

4.1 Cascade-structured neural networks

4.1.1 Introduction

In this section, we present an ensemble learning approach with neural networks to obtain the best possible performance for a 5-class classification on two datasets: KDD Cup 99 and NSL-KDD. The cascade-structured meta-specialists architecture is based on a three-step optimization method: 1/ data augmentation to rebalance the datasets; 2/ hyperparameters optimization to improve the performance of neural networks and 3/ ensemble learning to maximize the quality of detection. In order to show that our optimization method consistently gives better results than the state of the art, we compare it to other algorithms proposed in the literature.

4.1.2 Preprocessing and data augmentation

4.1.2.1 Datasets and preprocessing

For this work, we selected the two most popular datasets in intrusion detection: KDD Cup 99 and NSL-KDD. As stated previously, KDD Cup 99 has been repeatedly criticized by the scientific community for its deficiencies. While NSL-KDD corrects the redundancy and duplicating problem, it is based on the same 1998 DARPA Intrusion Detection Evaluation datasets. The connections and attacks in these datasets are therefore not a good representation of the activity and threats of a modern network. Moreover, the number of elements in the different classes of the two datasets is strongly imbalanced,

which favors the recognition of the most frequent classes. In addition, the distribution probabilities of classes vary significantly between the training set and the test set.

Nevertheless, we chose to study these datasets despite their shortcomings because they remain very popular within the field of intrusion detection research. They are important to us to better understand the intrusion detection field and how to improve them. Their popularity also facilitates the comparison between our results and those of other proposed solutions. However, the detection rates obtained on these datasets cannot be generalized to real networks, which present significantly different data from those of KDD Cup 99 and NSL-KDD. On the other hand, the proposed architecture is applicable to real computer networks. Comparing its results with those of the state of the art shows the viability of our solution.

In this work, the same preprocessing steps were applied for both datasets. First, the attack label of each malicious connection was transformed to one of the four attack classes. The values of the numerical features were then normalized between 0 and 1. Categorical features were finally one-hot encoded to be readable by the neural network [74].

4.1.2.2 Data augmentation

Learning from imbalanced data is a classical machine learning problem. The first step in our optimization process is to rebalance the training data, in order to obtain better results on the validation set. There are two ways to rebalance classes: under-sampling and over-sampling. As the names suggest, under-sampling reduces the populations of the most represented classes, while over-sampling increases those of the least represented classes. In our case, we want to decrease the number of connections in the normal and DoS classes, and increase those in the probe, R2L and U2R classes.

Indeed, the classifier must not overlearn the most represented classes to the disadvantage of the least represented classes. For instance, it would be easier for a classifier to completely ignore U2R attacks because of their very small proportion in the data set. Learning this class can reduce the detection rate of other classes, by adding a new possibility of misclassification for 99.99% of the KDD Cup 99 training set. On the other hand, ignoring U2R attacks would only cost 0.01% of the detection rate and avoid many errors. However, the purpose of this dataset is to correctly classify each class: the data augmentation process ensures that U2R attacks will not be ignored by the neural network.

16 data augmentation algorithms were used for this work. The following list provides a brief summary of each of them:

- Cluster centroids replaces a cluster from the majority class by the cluster centroid of a k-means model

- Random Under Sampler naively under samples the majority class by randomly removing data

- NearMiss [75] only keeps data from the majority class for which the average distance of the k nearest samples of the minority class is the smallest
- Edited Nearest Neighbours (ENN) [76] removes data whose class label differs from the class of at least half of its k-nearest neighbors
- Repeated Edited Nearest Neighbours [77] applies the ENN algorithm successively until it can remove no further points
- AllKNN [77] applies ENN several times and varies the number of nearest neighbours
- Instance Hardness Threshold [78] removes majority class samples that overlap the minority class sample space
- Random Over Sampling over samples the minority class by randomly duplicating data
- SMOTE (Synthetic Minority Over-Sampling TEchnique) [79] generates new samples by combining the data of the minority class with those of their close neighbours. It does not take into account neighboring examples that may come from other classes. Borderline 1 and 2 SMOTE and SVM SMOTE allow to find examples that may be hazardous.
- ADASYN [80] generates samples for minority classes examples that are the most difficult to learn using k-NN.
- SMOTEENN over samples the data using SMOTE and then cleans the result using ENN
- SMOTE Tomek over samples the data using SMOTE and then cleans the result using Tomek links [81]

We compared these data augmentation methods on the two datasets to determine which ones give the highest Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) or AUROC. Our classifier is a deep Multi-Layer Perceptron (MLP) with 3 hidden layers, each composed of 128 units with a Rectified Linear Unit (ReLU) activation function. It is trained on 20 epochs with a batch size of 256 with the Adam optimizer. The training is carried out on 80% of the training set, and the validation on the remaining 20% of the training set. The AUROC score presented in Table 4.1 is an average value obtained by repeating this process 10 times per method. We developed our code using Python 3 with the imbalanced-learn package [82] for data augmentation, and Tensorflow [83] with Keras [84] for the neural network.

We manually combined the best under-sampling method (Random Under Sampler) with the best over-sampling method (SVM SMOTE). This combination gave the best results for the two datasets, with a AUROC score of 1.0000 for normal class on NSL-KDD validation set (processed in 37 minutes and 22 seconds). The RUS + SVM SMOTE combination is used in the rest of the paper for both datasets.

Table 4.1: Comparison of data augmentation methods for NSL-KDD

Method	Normal class AUROC score
No sampling	0.9453
Under-sampling methods	
Cluster Centroids	0.9416
Random Under Sampler	0.9624
NearMiss	0.9375
Edited Nearest Neighbours	0.9607
Repeated Edited Nearest Neighbours	0.9609
Condensed Nearest Neigbour	0.8818
AllKNN	0.9618
Instance Hardness Threshold	0.7254
Over-sampling methods	
Random Over Sampling	0.9595
SMOTE	0.9533
Borderline 1 SMOTE	0.9524
Borderline 2 SMOTE	0.9559
SVM SMOTE	0.9693
ADASYN	0.9493
SMOTEENN	0.9561
SMOTE Tomek	0.9602

4.1.3 Hyperparameters optimization

Hyperparameters are the variables of a neural network set before training. This includes the number of neurons, batch size, optimizer, learning rate, activation functions, etc. Hyperparameters are often tuned manually, or by testing all possible combinations of a set of values. This technique, called grid search, has the advantage of necessarily finding the global optimum, i.e. the combination of parameters that offers the best classification results. However, it is difficult to use it in practice, due to the large number of parameters defined over extensive research groups. We chose two faster automated techniques: random search and Tree-structured Parzen Estimator (TPE) [85].

Random search randomly tests combinations of a range of values, with a fixed number of iterations. According to Bergstra and Bengio [86], random search can find better parameters than grid search in less computational time. The time allocated to this task is also easier to foresee, since the number of iterations is defined in advance. TPE is a Sequential Model-Based Optimization (SMBO) algorithm. Unlike random search, it chooses which parameters to test and converges to an optimal set of parameters. The choice of the optimization algorithm is data-dependent, that is why two different algorithms are tested.

This automated search is possible within short periods of time (a few minutes for each iteration) thanks to the size of the processed datasets: 743 MB for the entire KDD Cup 99 dataset, and 21.48 MB for NSL-KDD. These methods would be difficult to use with datasets of several GB. Optimization is a long process that is more reliable with few parameters. We manually defined as many parameters as possible that do not need to be optimized. For example, we know that the ReLU activation function [41] or the Adam optimizer outperforms the others. We also realized that any attempt at regularization, however slight, would lead to a decrease in accuracy. With this knowledge, the following hyperparameters have been optimized:

- Number of hidden layers: between 1 and 5 (step = 1);
- Number of units in each hidden layer: between 1 and 512 (step = 1);
- Number of epochs: between 1 and 200 (step = 1);
- Batch size = [16, 32, 64, 128, 256, 512, 1024, 2048];
- Adam's parameters: learning rate (between 10^{-5} and 0.2), beta 1 (between 0 and 1), beta 2 (between 0 and 1) and epsilon (between 10^{-9} and 10^{-5}).

We used the hyperopt library for its implementation of TPE [87]. Models have been trained on two GTX 1080 Ti GPUs.

Random search was quickly abandoned in favor of TPE, which consistently obtains better and faster results. Figure 4.1 shows the distribution of the best models (i.e., the top 20% of models in terms of accuracy) with the two methods. The search space corresponds to the set of values covered by the optimization algorithm (minimum and

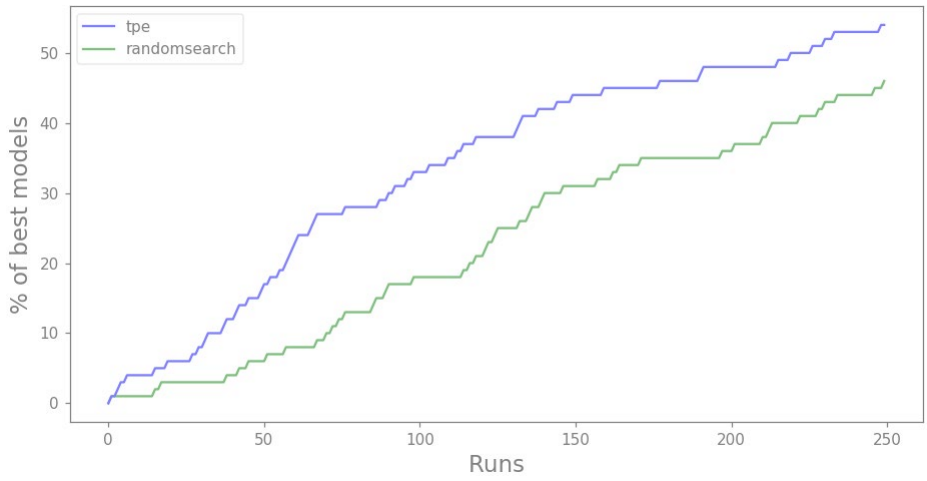

Figure 4.1: Comparison of results obtained with random search and TPE.

maximum values of neurons for instance). We started with large search spaces, which were gradually reduced manually by observing the best results to speed up the process.

Some hyperparameters clearly converged to an optimal value for both datasets. For example, a batch size of 256 obtained much better results than other values. Similarly, the number of layers quickly converged to a value of 1. KDD Cup 99 and NSL-KDD only have 41 features, far from the thousands of features found in image recognition. This may explain a lower need for generalization, and therefore a low number of layers. On the other hand, the numbers of units and epochs never clearly converged to a specific value. On average, they obtain better accuracy for values between 10 and 150, and between 5 and 60 for NSL-KDD respectively. Indeed, even after data augmentation, the distributions of classes in the training and test sets remain different: keeping a small number of epochs prevents overfitting. Likewise, the search of Adam's parameters values has been narrowed manually.

Several models have been built for each configuration of each dataset. In the first configuration, the model is trained with 80% of the training test, validated on 20% of the training test, and tested on the whole test set. In the second configuration, the model is trained on 60% of the training set, validated on 20% of the training set, and tested on the remaining 20% of the training set. This latter configuration gives better results, because the test set adds new attacks as well as a very different class distribution. It is used to compare our results with other solutions, but does not use the dataset as it was originally designed. The best models in each category are presented in Table 4.2.

4.1.4 Ensemble learning

4.1.4.1 Naive ensemble learning

Ensemble learning is a process combining several models to improve the overall predictive performance. This approach has been successful in many machine learning competitions,

Table 4.2: Classification accuracies for optimized neural networks on KDD Cup 99 and NSL-KDD.

Train set	Test set	Number of nodes	Number of epochs	Adam	Accuracy
NSL-KDD train (80%)	NSL-KDD test	81	15	lr: 0.075 beta1: 0.213 beta2: 0.850 ϵ: 9.50×10^{-6}	84.70%
NSL-KDD train (60%)	NSL-KDD train (20%)	125	35	lr: 0.128 beta1: 0.125 beta2: 0.958 ϵ: 2.42×10^{-6}	99.29%
KDD Cup 99 train (80%)	KDD Cup 99 test	68	32	lr: 0.001 beta1: 0.9 beta2: 0.999 ϵ: 10^{-8}	93.77%
KDD Cup 99 train (60%)	KDD Cup 99 train (20%)	130	24	lr: 0.001 beta1: 0.9 beta2: 0.999 ϵ: 10^{-8}	99.95%

such as KDD Cup 2009. The general idea is that a combination of weak learners is more effective than a single strong learner, thus increasing the accuracy of the model. Ensemble helps to reduce variance and bias, which are the main causes of difference between predicted and real values.

We tested the performance of this approach by naively combining our two best models from the previous step. There are several combination rules to create an ensemble classifier: averaging their predictions, keeping only the maximum value, adding them up, multiplying them, etc. [88] We tested in Table 4.3 different algebraic combiners to create the final prediction p from p_1 (model 1 with 84.70% classification accuracy on NSL-KDD test set) and p_2 (model 2 with 84.17% classification accuracy on NSL-KDD test set).

All combination rules work better than the best model of the previous section, especially the mean and sum rules with a +0.18% increase in accuracy. This increase can be explained by the way classifiers make their predictions. On average, a classifier has less confidence in its predictions when they turn out to be false than when they are true. Combining a false prediction with a true prediction thus favors the latter, as long as the classifiers do not make too many mistakes.

Table 4.3: Comparison of different combination rules for ensemble learning on NSL-KDD test set.

Combination rule	Prediction p for N models	Accuracy
Mean rule	$p = \frac{1}{N}\sum_{i=1}^{N} p_i$	84.88%
Maximum rule	$p = max_{i=1,...,N}\{p_i\}$	84.82%
Sum rule	$p = \sum_{i=1}^{N} p_i$	84.88%
Product rule	$p = \prod_{i=1}^{N} p_i$	84.78%

4.1.4.2 Meta-specialists for ensemble learning

This statement led us to create classifiers specialized in the detection of a single class. There are five types of specialists, one for each class of the dataset. These specialists can be 5-class (normal, DoS, probe, R2L, U2R) or 2-class (their class or not) classifiers. We tested both approaches and obtained better results for 5-class specialists on normal classes, DoS, R2L and U2R (but not probe), that is why we continue to use 5-class classifiers in the following.

We applied the same method as in the first and second sections for the training of these specialists. The preprocessing of the training set depends on the classifier's specialty. Indeed, the class in which the classifier is specialized is over-represented (1:5 to 1:30) compared to the others. First, all other classes are under-sampled with the Random Under Sampler. If the specialty of the classifier is probe, R2L or U2R, this class is then over-sampled around 20,000 connections. The specialist's neural network is optimized with the TPE using the process described previously. In addition to the hyperparameters, the class ratio is also optimized by the TPE on a validation set.

We then applied the ensemble learning method to each of the 5 sets of specialists. However, models have different accuracies: some perform better than the others on a dataset. Increasing the contribution of the best models in the final prediction would naturally lead to better results. But poor models should not be systematically excluded from the ensemble. They can indeed be specialized in rare forms of connections that the best models do not recognize. This way of favoring the best models can be implemented by adding weights λ_i to the prediction p_i of each model i in the previous combination rules. These weights are then optimized with the TPE on a validation set to maximize the AUROC score of the meta-specialist. In addition to the previous combination rules, we added majority voting, which selects the class that receives the largest total votes. We thus defined a meta-specialist as the composition of several specialists from the same class, as a group that only participates in the classification of its specialty. Results with meta-specialists for NSL-KDD are shown in Table 4.4. Mean rule achieved the same

accuracy than sum and product rules but is faster to compute (approximately 1 hour and 20 minutes, depending largely on models). This is why mean rule is used for both datasets in the following.

Table 4.4: Comparison of different combination rules with meta-specialists for ensemble learning on NSL-KDD test set.

Combination rule	Prediction p of a meta-specialist for N models	Accuracy
Mean rule	$p = \frac{1}{N} \sum_{i=1}^{N} \lambda_i p_i$	86.33%
Maximum rule	$p = max_{i=1,\dots,N}\{\lambda_i p_i\}$	86.01%
Sum rule	$p = \sum_{i=1}^{N} \lambda_i p_i$	86.33%
Product rule	$p = \prod_{i=1}^{N} \lambda_i p_i$	86.33%
Majority voting	$p = max_{i=1,\dots,N} \sum_{i=1}^{N} \lambda_i p_i$	86.17%

4.1.4.3 Cascade-structured meta-specialists architecture

A bias can be easily detected by looking at the results of each meta-specialist independently. Indeed, meta-specialists tend to over-recognize their own specialty in the connections presented to them. This is a side effect of their training where they have seen their own class more than others. This bias is a problem for rare and therefore unreliable attacks like R2L and U2R, which can produce many false positives. A solution to mitigate it is to create a specific architecture, where the non-classified connections are successively presented to the different meta-specialists, as shown in Figure 4.2.

In this architecture, the entire NSL-KDD dataset is first presented to the normal meta-specialist. This classifier only classifies normal connections. Connections flagged as "normal" are subtracted to the dataset, which is then presented to the probe meta-specialist. This process is repeated for R2L, DoS and U2R meta-specialists. The order of meta-specialists was determined by selecting the one that gave the best AUROC score on the validation set. All remaining connections, those that have not been recognized by any meta-specialist, are then classified. The class of each of these connections is determined by the specialist who gives them the highest probability.

Unlike naive ensemble learning models, specialists have never been trained or validated on the test set, in order to avoid data leakage. Their performance was measured on a validation set (20% of the training set), despite its important differences with the test set. Indeed, validating the weight optimization of specialists on the test set would greatly improve the results. Under these conditions, this architecture achieves 92.66% classification accuracy on the NSL-KDD test set. Table 4.5 presents the final perfor-

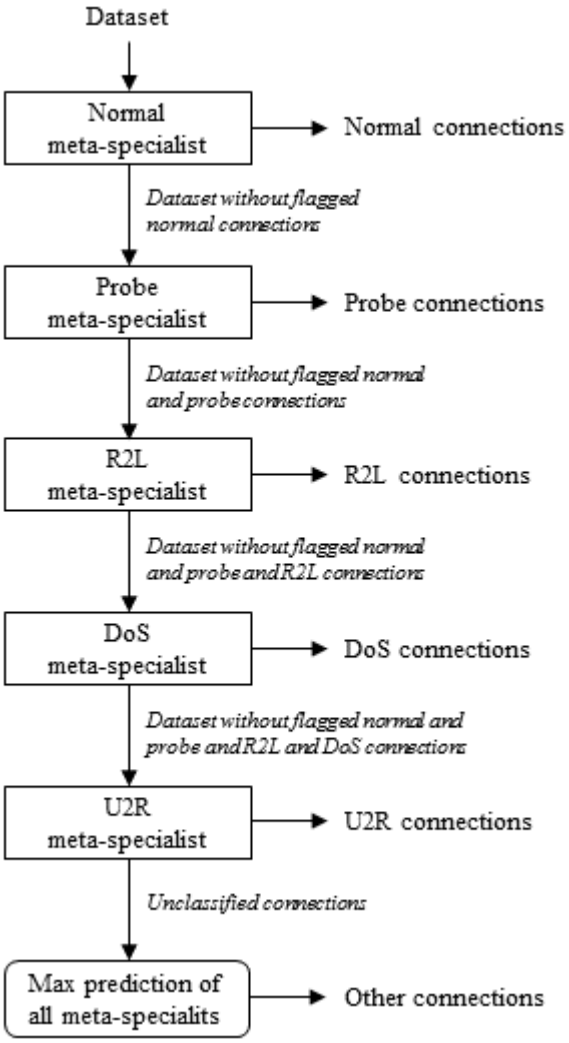

Figure 4.2: Cascade-structured meta-specialists architecture for NSL-KDD.

mance for our architecture on KDD Cup 99 (with max rule combination) and NSL-KDD (with sum rule combination).

Table 4.5: Classification accuracies for cascade-structured meta-specialists architecture on KDD Cup 99 and NSL-KDD.

Train set	Test set	Accuracy
NSL-KDD train (80%)	NSL-KDD test	88.39%
NSL-KDD train (60%)	NSL-KDD train (20%)	99.91%
KDD Cup 99 train (80%)	KDD Cup 99 test	94.44%
KDD Cup 99 train (60%)	KDD Cup 99 train (20%)	99.95%

The classification performance of each class within the architecture is detailed in Tables 4.6 and 4.7.

Table 4.6: Summary of test results for cascade-structured meta-specialists architectures for KDD Cup 99 (classification accuracy = 94.44%).

	Normal	DoS	Probe	R2L	U2R
Accuracy	98.10%	94.77%	99.73%	96.32%	99.97%
TPR	97.54%	98.74%	90.06%	36.28%	28.57%
FPR	0.33%	6.19%	0.14%	0.35%	0.01%
F1 score	0.9870	0.8803	0.8986	0.5089	0.2985
AUROC	0.9861	0.9628	0.9496	0.6797	0.6428

Table 4.7: Summary of test results for cascade-structured meta-specialists architectures for NSL-KDD (classification accuracy = 88.39%).

	Normal	DoS	Probe	R2L	U2R
Accuracy	95.02%	92.37%	96.86%	93.40%	99.12%
TPR	88.87%	96.10%	85.38%	64.92%	35.82%
FPR	1.94%	10.46%	1.75%	2.42%	0.69%
F1 score	0.9220	0.9156	0.8540	0.7158	0.1943
AUROC	0.9347	0.9282	0.9181	0.8125	0.6756

Finally, Table 4.8 shows a comparison study on NSL-KDD between our model and previous results in terms of classification accuracy and FPR. Our solution performs better than the best 2-class classifiers in the literature in both metrics.

4.1.5 Conclusion

We presented a three-step methodology for optimizing intrusion detection with neural networks. The cascade-structured meta-specialists architecture is based on the creation of specialized classifiers in a single class. Specialists are first trained on a modified

Table 4.8: Comparison study on NSL-KDD.

Study	Accuracy	FPR
Our solution	88.39%	1.94%
Two-level classifier ensemble [89]	85.016%	12.6%
Bagging (J48) + feature selection [90]	84.25%	2.79%
GAR-forest + feature selection [91]	85.05%	12.2%
SVM + feature selection [92]	82.37%	15%

training set to over-represent their class. The hyperparameters of these classifiers are then optimized to maximize their accuracy on a validation set. Specialists of the same class are combined into a meta-specialist. Non-flagged connections in the dataset are successively tested by all meta-specialists. This system has proven to greatly improve the quality of detection on KDD Cup 99 and particularly on NSL-KDD, with a classification accuracy of 88.39% and 1.94% FPR. It could be applied to any other labeled dataset for intrusion detection, with a similar performance increase compared to a naive classifier.

This approach could be improved by combining neural networks with other machine learning algorithms (e.g., Random Forest or SVM). These algorithms are more deterministic than neural networks, and could thus compensate for certain deficiencies of the latter. Besides, preprocessing is done on the entire training dataset, but selecting a combination of data augmentation algorithms class by class would make more sense. This would help to extend the classification system to the attacks themselves rather than the categories.

4.2 Ensemble of machine learning techniques

4.2.1 Introduction

Previous work used only one machine learning model: the neural network (MLP). It is possible to better exploit the contribution of the ensemble learning by adding other types of models, with different bias-variance tradeoffs. The data augmentation process is also being redesigned, by selecting one algorithm per class and no longer one for the entire dataset. This method better represents the differences between very sparsely populated (U2R) and slightly under-represented (probe) classes. A new architecture without cascading meta-specialists is also proposed. Finally, the entire training and optimization process (selection of algorithms and coefficients) is fully automated.

4.2.2 Dataset et preprocessing

The KDD Cup 99 dataset has been abandoned compared to the latest work to speed up the training process. KDD Cup 99 is larger than NSL-KDD, which makes it longer to process. Its data is also less relevant because of the high number of redundant

connections and several incorrect records. This work is therefore entirely based on NSL-KDD, but the same architecture can be applied to KDD Cup 99.

NSL-KDD suffers from the problem of unbalanced classes, which requires a step of data augmentation. This step is modified here to take into account the specificities of each class. Indeed, even two poorly represented attacks do not contain the same data. It is therefore likely that the best over-sampling algorithm differs between these two classes. We want to create 5 training datasets: one specialized in each class. The specialized class is over-sampled, while the others are under-sampled with a ratio of 1:10. This specific ratio was determined using the results of previous work (it was defined between 1:5 and 1:30). The classifier who learns on these specialized datasets will become a specialist of the most represented class.

Two algorithms are therefore needed for each class: one for under-sampling and one for over-sampling. The best under-sampling algorithm is combined with the best over-sampling algorithm to form a specialized dataset for a class with a 1:10 ratio. A naive classifier is used to determine the performance of each algorithm for each class. A MLP composed of a single hidden layer of 128 neurons was chosen for this purpose. It uses ReLU as the activation function, and the Adam optimizer. The AUROC of the specialized class is used as a metric to measure the performance of our classifier. This neural network is trained on 80% of the training set, and the area under the ROC curve is measured on the remaining 20%.

Table 4.9 shows the results of this data augmentation step, including the best under-sampling and over-sampling methods for each class: AllKNN + SVM SMOTE for normal, Repeated Edited Nearest Neighbours + ADASYN for DoS, One Sided Selection + SMOTEENN for probe, Random Under Sample + Borderline 1 SMOTE for R2L, and Cluster Centroids + SMOTE Tomek for U2R. These two methods are combined as described above to form the a training set for each class.

4.2.3 Training

Different machine learning algorithms for classification are used: decision tree, random forest, extra tree, extra trees, k-Nearest Neighbors (KNN), SVM, logistic regression, Naive Bayes, gradient boosting and multilayer perceptron (MLP). These classifiers are from the sklearn library [14], except for the MLP created with Keras [84] and Tensorflow [15].

Each algorithm has a set of parameters whose values influence the classification performance. Optimizing these parameters is therefore an important step in improving detection accuracy. Several optimization methods are commonly used in machine learning. The most common one involves an expert who chooses the most relevant parameters for the problem being studied. This method requires a good knowledge of the algorithm to be optimized as well as the impact of the different parameters. It would be possible to set the best parameters for the dataset studied (NSL-KDD) here. But in order to propose a method that can be transposed to other data, we want to automate the parameter search.

We continue to use the TPE as a parameter optimization algorithm here. The

Table 4.9: Comparison of data augmentation methods for each class of NSL-KDD

Method	AUROC				
	Normal	DoS	Probe	R2L	U2R
No sampling	0.9488	0.8930	0.9158	0.7429	0.9605
Under-sampling methods					
Cluster Centroids	0.9378	0.8915	0.9181	0.8548	0.9739
Random Under Sampler	0.9616	0.9000	0.9338	0.8674	0.9715
NearMiss	0.9377	0.7491	0.8691	0.5929	0.2762
Edited Nearest Neighbours	0.9382	0.9071	0.9234	0.7932	0.9575
Repeated ENN	0.9614	0.9162	0.9081	0.7773	0.9626
Condensed Nearest Neigbour	0.9022	0.6521	0.8518	0.6224	0.9174
One Sided Selection	0.9415	0.6498	0.9543	0.6986	0.9435
AllKNN	0.9659	0.9019	0.9138	0.8012	0.9701
Instance Hardness Threshold	0.6944	0.9079	0.8753	0.8104	0.9458
Over-sampling methods					
Random Over Sampling	0.9611	0.9083	0.9364	0.7838	0.9714
SMOTE	0.9423	0.8965	0.8918	0.8025	0.9511
Borderline 1 SMOTE	0.9597	0.9133	0.9173	0.8535	0.9675
Borderline 2 SMOTE	0.9542	0.9146	0.9125	0.7659	0.9601
SVM SMOTE	0.9644	0.8924	0.9273	0.7990	0.9696
ADASYN	0.9485	0.9188	0.9324	0.7855	0.9717
SMOTEENN	0.9500	0.8742	0.9457	0.7365	0.9560
SMOTE Tomek	0.9627	0.8784	0.9178	0.7420	0.9799

AUROC is also used to evaluate the classification performance. Each machine learning algorithm is trained on 80% of the training set of each class, obtained in the previous section. Its performance is then evaluated on the remaining 20% from the original NSL-KDD training set (i.e. without the class specific sampling). The goal of TPE is to find, for each classifier, the parameters that maximize the area under the curve calculated on this 20%. These parameters are saved for each algorithm applied to the training set of each class, and will be reused in the following.

4.2.4 Ensemble learning

Ensemble learning is used in a different way than previously. We want to create a set for each class, combining the best classifiers of this class into one ensemble. Indeed, trained models have different detection performance, some being more accurate than others. Adding inaccurate models leads to an accumulation of errors. A selection of the best models is used to prevent this deterioration in the quality of detection.

Some algorithms are more effective on certain classes by design, which is why this selection process is important to choose the best models for every class. One way to approach this issue is to see it as a tradeoff between bias and variance. Some classifiers are more effective on data with high bias and low variance (e. g. Naive Bayes), while others are more effective on data with low bias and high variance (e. g. KNN). For small datasets, algorithms in the first category are generally preferred, while others are more suitable for large datasets.

The selection of classifiers is often done manually, or by correlating the results of each algorithm to select only the least correlated ones. An innovative automatic method is adopted here, allowing a flexible weighting coefficient to be assigned to the predictions of each algorithm in the set. The models are first ranked from best to worst, based on their area under the ROC curve of their class on the validation set. The predictions p_1 of the worst model and p_2 of the second worst model are combined by a weighted average (equation 4.1). The TPE then tries to optimize the ratios λ_1 and λ_2 of the average in order to obtain the best possible accuracy on this set. These ratios are kept and the process is repeated between the formed set, and the new algorithm that we are trying to add. If there is no ratio to improve the accuracy of the set, the new algorithm is ignored and the next one is tested instead. The choice to start with the worst models is explained by the gradual decrease in the ratio of the first models as new ones are added.

$$p = \sum_{i=1}^{N} \lambda_i p_i \tag{4.1}$$

This method results in the creation of 5 ensembles, each specialized in a class of NSL-KDD. Each ensemble produces a prediction p, which is the weighted sum of the p_i predictions of each algorithm i by a ratio λ_i. Overfitting is mitigated by adding the worst models at the beginning of the process, which however contribute to the classification.

These ensembles are then combined to obtain the final classification on the test set. We choose the ensemble that assigns the highest probability to its class to determine

the final class of each connection. This combination rule is based on the idea that an erroneous prediction will most of the time have a lower probability than a correct prediction. With this method we obtain a global accuracy (Jaccard index) on the 5 classes of the NSL-KDD test set of 86.59%. The rate of false positives (normal traffic classified as an attack) also remains low, at 1.66%. The overall results obtained for each class are detailed in Table 4.10.

Table 4.10: Summary of test results on NSL-KDD test set

	Normal	DoS	Probe	R2L	U2R
Accuracy	95.20%	90.91%	95.87%	91.73%	99.48%
TVP	88.85%	95.99%	86.70%	50.02%	43.28%
TFP	1.66%	12.94%	3.03%	2.15%	0.36%
F1 score	0.9245	0.9009	0.8185	0.6076	0.3295
AUROC	0.9359	0.9153	0.9184	0.7394	0.7146

4.2.5 Conclusion

We presented a method for automatically optimizing a set of machine learning algorithms for intrusion detection. First, the training set is preprocessed in order to generate a specialized dataset for each class. A collection of machine learning algorithms are trained (and, in some cases, optimized) for each of these datasets. Finally, the predictions of the best algorithms are combined by optimizing their weights to improve detection accuracy.

This method offers a high accuracy of 86.59% on NSL-KDD with a low FPR (1.66%). These scores are relatively similar to those obtained with the method detailed in the previous section (88.39% accuracy and 1.94% FPR). Added machine learning algorithms generally perform worse than neural networks. Indeed, neural networks have been selected among the best performing models for each of the 5 classes. Moreover, our new data augmentation method does not significantly improve the detection of rare attacks. The R2L attacks' AUROC score drops from 0.8125 to 0.7394 and the U2R attacks' AUROC score increases from 0.6756 to 0.7146. Adding weights to increase the cost of errors on the rarest attacks would be another approach to improve their detection.

This method allows to consider an automatic deployment of an IDS solution based on machine learning. However, NSL-KDD is not a good representation of a real network traffic. This method is based on supervised learning, and therefore requires a labelled training set. For a real deployment, it should be based on a pre-established dataset, or an automatic labelling method. Our process could also be improved by selecting the best features for each algorithm. This would remove the least important parameters from the dataset to reduce its dimensionality. This reduction would shorten the computation time, and increase the accuracy of detection, especially for machine learning algorithms that are highly subject to the curse of dimensionality.

The next chapter will consider more realistic network traffic. Techniques from supervised learning are explored to overcome the shortcomings of this type of training, while

maintaining its advantages in terms of detection quality.

Chapter 5

Supervised Techniques to Improve Intrusion Detection

This chapter presents techniques to improve intrusion detection. These techniques are based on IDSs but are not themselves IDSs. Section 1 introduces the generation of signatures from anomalies for misuse detection. Section 2 presents applications of transfer learning in order to solve certain problems with supervised learning models for intrusion detection.

5.1 Anomaly to signature

The ability of anomaly-based IDSs to detect unknown attacks is their greatest asset. However, an attack identified by an anomaly-based IDS cannot be reliably detected every time: subtle changes in the header parameters can completely modify its prediction. Signature-based IDS are much more reliable in attack detection, since each attack corresponds to one or more well-defined signatures. This correspondence can be used to explain why a packet is considered as an intrusion, since the signature specifies the parameters that led to this classification. It is also possible to trace the name or the type of the attack, which can be useful for reaction mechanisms (IPS) after detection.

The ability to transform an anomaly into a signature that can be interpreted by a signature-based IDS is therefore very valuable in several situations. First, a hybrid system composed of both solutions can self-reinforce using this principle, feeding its own signature database with the anomaly detection component. Moreover, signature-based IDSs generally require less computational power than anomaly-based IDSs using machine learning models. Implementing signature-based IDSs in constrained environment like embedded systems is therefore a good solution that can be improved with anomaly detection. Finally, signature-based IDSs represent a large majority of the already deployed IDSs worldwide. Feeding their signature databases with anomaly detection would improve this detection method (quick updates), without completely changing software.

There are three popular open-sourced signature-based IDSs: Snort, Suricata, and Zeek. Snort was released in 1998 and is the most used of the three. It can be configured

in three different modes: packet sniffer (displays them on the console), packet logger (writes logs to the disk), and network intrusion detection system. In this last mode, the network traffic is compared to a rule set defined by the administrator. These rules also contain Snort's reactions in case of successful detection. Suricata's architecture is different than Snort, but they can be used in the same way, with the same signatures. Suricata was released in 2009 and compensates for its relative youth with multi-threading capabilities. Finally, Zeek (formerly Bro) can also be used as an anomaly-based IDS. The analysis engine converts network traffic into a series of events. Zeek does not use the same signatures, but has its own script language. These scripts are then applied to the network events for intrusion detection or other various tasks (notifications, additional analysis, etc.).

We chose to work with Snort signatures because of their portability to Suricata, their relatively simple semantics, and Snort's popularity. Figure 5.1 shows an example with a Snort rule to detect SYN packets on a local network.

alert tcp any any -> 192.168.0.1/24 any (flags: S; msg: "SYN packet";)

Figure 5.1: Sample Snort rule.

Figure 5.2 summarizes the semantics of Snort signatures.

<action> <protocol> <source> <direction> <destination> (<options>)

Figure 5.2: Snort rules syntax.

The first field in a rule is the action Snort will take if a packet with all the attributes indicated in the rule is detected. There are 6 different rule actions:

- alert – generate an alert using the selected alert method, and then log the packet;
- log – log the packet;
- pass – ignore the packet;
- drop – block and log the packet;
- reject – block the packet, log it, and then send a TCP reset if the protocol is TCP or an ICMP port unreachable message if the protocol is UDP;
- sdrop – block the packet but do not log it.

Snort is capable of analyzing five network protocols: IP, TCP, UDP, ICMP, HTTP (or any). "Source" and "destination" describe respectively the IP and the port from which traffic is coming, and the IP and port on which traffic is coming for establishing the connection. The direction ("->" or "<>") indicates the direction of the traffic between sender and receiver networks. Several rule options can be set to refine the detection process:

- msg: prints the content of the message with the packet dump or alert;
- reference: includes references to external attack identification systems;
- gid: identifies what part of the IDS generates the event when a specific rule is activated;
- sid: uniquely identifies the IDS rules;
- rev: uniquely identifies revisions of the IDS rules;
- classtype: categorizes a rule as detecting a subtype of a more general attack class;
- priority: assigns a severity level to rules;
- metadata: embeds additional information about the rule.

It is necessary to correctly detect the attack for which we want to obtain a signature. Several machine learning algorithms are evaluated to achieve the most accurate classification of different attacks. These algorithms predict an anomaly score between 0 and 1 rather than a direct classification. This method allows us to modulate the number of false positives using a threshold. The dataset CSE-CIC-IDS2018 is used for this work. It has a large number of connections and very high classification rate for all machine learning models. The decision tree is selected for its excellent performance (100% TPR, 0% FPR) and its low training time (11.4s on the DDoS set).

The decision tree has a particular architecture that allows to understand the choices made by the algorithm for its classification. Each node corresponds to an if...else condition on a parameter of the dataset. Figure 5.3 shows a plot of the conditions at the top of the decision tree:

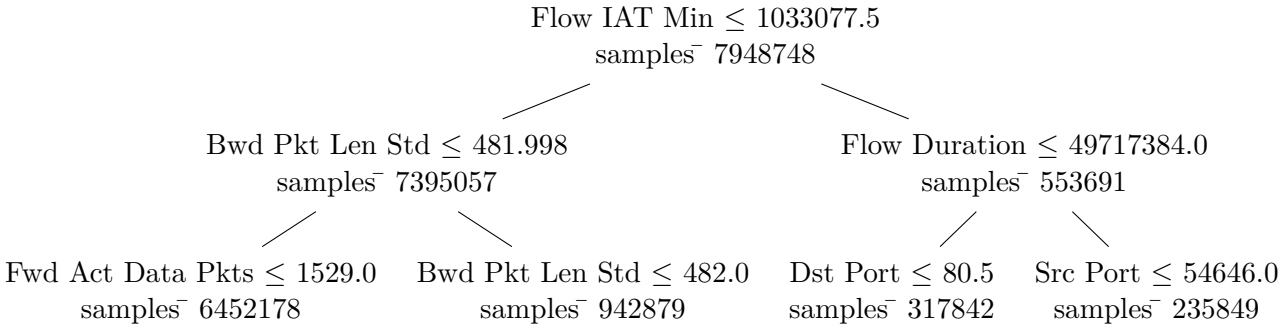

Figure 5.3: Three first levels of the decision tree.

The total DDoS attack classification tree of CSE-CIC-IDS2018 has a depth of 18 and 333 leaves. It is possible to convert these rules into a succession of hard-coded conditions. This solution could replace an IDS entirely, and instead call a function to

Algorithm 1: Rules extracted from the decision tree.

```
Result: anomaly scores
initialization;
if Flow Pkts/s ≤ 1033077.5 then
    if Bwd Pkt Len Std ≤ 481.998 then
        if Subflow Bwd Pkts ≤ 1529.0 then
            ...
        else
            ...
        end
    else
        if Bwd Pkt Len Std ≤ 482.0 then
            ...
        else
            ...
        end
    end
else
    if Flow Duration ≤ 49717384.0 then
        if Dst Port ≤ 80.5 then
            ...
        else
            ...
        end
    else
        if Src Port ≤ 54646.0 then
            ...
        else
            ...
        end
    end
end
```

analyze each packet and determine its threat level. The algorithm 1 shows an example of the conversion of the decision tree above into Python code.

The decision tree allows us to find the features that contributed most to the classification. Knowing the contribution of each feature allows to retrieve information on the interest to create (feature engineering) or to collect more parameters if it is possible. It also helps to remove the features that are the least useful for classification, thus reducing the size of the dataset and shortening the training time of the model. The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. It is also known as the Gini importance [93]. Table 5.1 shows the most important features to determine that a flow is an attack for this decision tree (importance $> 10^{-5}$). 51 features among the 76 of the CSE-CIC-IDS2018 dataset have a null importance and 7 of them have an importance less than 10^{-5}.

Table 5.1: Feature importances for DDoS detection on CSE-CIC-IDS2018 (importance $> 10^{-5}$)

#	Feature	Importance
1	Bwd Pkt Len Mean	5.19e-01
2	Flow IAT Mean	2.08e-01
3	Timestamp	1.71e-01
4	Protocol	6.43e-02
5	Subflow Fwd Byts	9.01e-03
6	SYN Flag Cnt	8.59e-03
7	Idle Mean	7.63e-03
8	Bwd URG Flags	2.18e-03
9	Subflow Bwd Byts	1.47e-03
10	Active Std	1.08e-03
11	Bwd Pkt Len Std	1.93e-04
12	RST Flag Cnt	1.42e-04
13	Flow IAT Std	5.70e-05
14	Bwd PSH Flags	1.97e-05
15	Fwd Pkts/s	1.75e-05
16	ECE Flag Cnt	1.59e-05
17	Bwd Blk Rate Avg	1.58e-05

Snort signature generation does not require as much information, nor a model as explainable as a decision tree. In fact, it can be done with any machine learning algorithm, since all the information needed for generation are included in the dataset features. The generation thus requires to detect a flow as an attack, then to retroactively search the different features that constitute this flow to generate a signature. Twelve features are used among the 76 of the CSE-CIC-IDS2018 dataset: : Protocol, Src IP, Src Port, Dst IP, Dst Port, ACK Flag Cnt, SYN Flag Cnt, PSH Flag Cnt, RST Flag Cnt, FIN Flag Cnt, URG Flag Cnt, et Label. Figure 5.4 shows examples of Snort signatures generated

with this method.

```
alert tcp 52.14.136.135 50819 -> 172.31.69.25 80 (msg:“DDoS attacks-LOIC-HTTP”;
                        flags:PR; classtype: denial-of-service;)
alert tcp 52.14.136.135 50820 -> 172.31.69.25 80 (msg:“DDoS attacks-LOIC-HTTP”;
                        flags:PR; classtype: denial-of-service;)
alert tcp 52.14.136.135 50821 -> 172.31.69.25 80 (msg:“DDoS attacks-LOIC-HTTP”;
                        flags:PR; classtype: denial-of-service;)
alert tcp 52.14.136.135 50822 -> 172.31.69.25 80 (msg:“DDoS attacks-LOIC-HTTP”;
                        flags:PR; classtype: denial-of-service;)
```

Figure 5.4: Generated Snort rules.

The output of this algorithm is verbose, since it generates a signature for each flow detected as an attack. It is possible to compress this output using Snort's ability to handle ranges of address and port numbers. This compression can be performed on four parameters: source address, source port, destination address, and destination port. Figure 5.5 shows the same example as above, but with the source ports aggregated to give the following signature:

```
alert tcp 52.14.136.135 50819-50822 -> 172.31.69.25 80 (msg:“DDoS
attacks-LOIC-HTTP”; flags:PR; classtype: denial-of-service;)
```

Figure 5.5: Compressed Snort rule.

These signatures can be incorporated directly into the Snort database via updates. They can also be easily generated by the anomaly detection component of a hybrid system. We did not implement these rules in Snort because it would have been necessary to replay the entire traffic to test them. Since the correspondence between the flows and the packets in the CSE-CIC-IDS2018 dataset was not direct (flows are an aggregation of packets and lose some information), this work was considered too costly for the results it would have provided.

Indeed, the semantics of Snort rules is too simple to fully exploit the possibilities offered by the conversion of anomalies into signatures. Future work could focus on code generation using Zeek's language, which would allow more accurate simulation of the detected anomaly. This conversion method is easily implemented in real use cases, but requires a low false positive rate from the anomaly-based IDS to be really useful.

5.2 Transfer learning

The main problem with algorithms trained in a supervised manner is the need for a very specific kind of dataset. These datasets must be labeled, contain both normal and attack traffic, and be located on the target network that the IDS must protect. The process of creating such a dataset is a complex task, which requires a lot of planning work on many aspects: choice of attacks and schedules, automatic labeling, traffic representativeness,

etc. The Canadian Institute for Cybersecurity points out that such a dataset is very rare to obtain in reality [26].

Transfer learning is a machine learning method focused on re-using a model developed for a task A as the starting point for a model on a related task B. For instance, a convolutional neural network designed to detect cats can be re-used to detect dogs instead. In our use case, tasks A and B correspond to two different computer networks. The main benefits of transfer learning are that it can speed up the training process and obtains better performance than a model simply developed for task B [94].

The use of transfer learning allows the training to be based on a set of data that is not exactly representative of the monitored traffic. The ability of the model to adapt to a new context is not only useful for its deployment. This characteristic remains important so that it can continue to adapt to the changes that will inevitably occur in network traffic (addition or removal of systems, new protocols, etc.). This generability is therefore useful both for training the model and throughout its use.

The CICIDS2017 and CSE-CIC-IDS2018 datasets are good candidates for testing the transfer learning capabilities of detection algorithms. Indeed, both datasets are based on flows retrieved with CICFlowMeter and share common attacks: DoS, web attacks, botnet, brute-force. However, the topology of the monitored network is very different, which represents a typical scenario where the IDS is trained on a dataset that does not correspond to the network on which it is deployed.

The gain in generability of these models is accompanied by a minimization of the importance of the network topology for intrusion detection. Network-agnostic IDSs are useful for their ability to be deployed on any network, but their performance would be inferior to IDSs specifically trained to understand the behavior of a particular network. This approach is therefore a trade-off towards ease of deployment, in exchange for probable loss of detection performance.

5.2.1 Datasets and preprocessing

The objective is to train a machine learning model on one dataset, then test it on the second one. Good performance on the test set would indicate a good ability of the model to ignore the network topology for attack detection. Since CSE-CIC-IDS2018 has a much higher number of flows than CICIDS2017, it was chosen as a training set. This is consistent with the fact that the training set is usually larger than the test set, with typical ratios such as 80/20 or 64/33. 20% of CSE-CIC-IDS2018 is used as a validation set to test the performance of machine learning algorithms on their original network topology.

Although both datasets are provided by the Canadian Institute for Cybersecurity using the same tools, the two datasets have some differences that need to be corrected in order to be used together.

First of all, the categories of attacks are not organized in the same way. CICIDS2017 counts 8 categories: benign, brute-force, DoS, web attacks, infiltration, botnet, port scan, and DDoS. CIC-CSE-2018 has only 6: brute-force, DoS, web attacks, infiltration, botnet, and DDoS. This difference can be explained by the absence of a day without

any attacks for CIC-CSE-2018 and the merging of port scans with DDoS attacks. These two types of attacks are therefore not considered in the rest of this study, as they are deemed too different in their implementation. Similarly, the infiltration attack exploits the downloading of a file on Dropbox in both cases, but with a process too different to consider that it is the same attack. The four categories of intrusions retained are DoS, web attacks, botnet, and brute-force.

Both datasets also require the same features. The feature 'Fwd Header Length.1' has therefore been removed from CICIDS2017, and the features 'Protocol' and 'Timestamp' have been removed from CSE-CIC-2018. Format problems despite data conversion also led us to remove the features 'Flow Bytes/s' and 'Flow Packets/s' from both datasets (named 'Flow Byts/s' and 'Flow Pkts/s' in CSE-CIC-IDS2018). After this harmonization step, both datasets have 75 features + 1 label. These features are normalized between 0 and 1 with a min-max scaler.

5.2.2 Comparison of different machine learning models

Seven machine learning algorithms have been selected to test their performance in transfer learning: logistic regression, decision tree (Table 5.3), random forest, extra tree, Gaussian Naive Bayes, KNN, and MLP.

Table 5.2: Transfer learning results for logistic regression.

Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CSE-CIC-IDS2018 (training)	DoS	99.93%	99.94%	0.07%	0.999	0.999
	Web atk.	99.97%	27.98%	0.00%	0.640	0.437
	Brute-force	99.95%	100.00%	0.08%	1.000	0.014
	Botnet	96.16%	99.91%	5.25%	0.973	0.934
Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CICIDS2017 (test)	DoS	71.89%	68.54%	26.19%	0.712	0.640
	Web atk.	91.16%	4.95%	7.72%	0.486	0.014
	Brute-force	3.08%	100.00%	100.00%	0.500	0.060
	Botnet	93.58%	0.88%	5.46%	0.477	0.003

The results of these different learning machine models vary greatly. The classification of some models on the test set is barely better than random (Gaussian Naive Bayes, KNN, MLP), while others perform well on some classes (decision tree, random forest). The attack categories are themselves more or less difficult to classify correctly: DoS and brute-force attacks are the easiest to detect, while web attacks are difficult for all classifiers (the best AUROC is 0.519, barely better than random).

The best classifier of all the models tested is the decision tree, with excellent performance on brute-force attacks (AUROC = 0.996). Only about half of the denial of service attacks are detected (TPR = 46.08%), with a high false positive rate (1.30%). Web attacks and botnets do not get sufficient results to be considered detected.

Table 5.3: Transfer learning results for decision tree.

Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CSE-CIC-IDS2018 (training)	DoS	100.00%	100.00%	0.00%	1.000	1.000
	Web atk.	99.99%	87.50%	0.00%	0.937	0.926
	Brute-force	100.00%	100.00%	0.00%	1.000	1.000
	Botnet	100.00%	99.99%	0.00%	1.000	1.000
Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CICIDS2017 (test)	DoS	79.51%	46.08%	1.30%	0.724	0.621
	Web atk.	98.42%	4.22%	0.33%	0.519	0.066
	Brute-force	99.29%	99.99%	0.73%	0.996	0.898
	Botnet	99.22%	25.02%	0.01%	0.625	0.396

Table 5.4: Transfer learning results for random forest.

Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CSE-CIC-IDS2018 (training)	DoS	91.09%	90.68%	8.72%	0.910	0.864
	Web atk.	99.97%	29.18%	0.00%	0.646	0.452
	Brute-force	99.96%	100.00%	0.07%	1.000	0.999
	Botnet	85.91%	48.71%	0.02%	0.743	0.655
Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CICIDS2017 (test)	DoS	68.11%	26.06%	7.72%	0.592	0.374
	Web atk.	98.70%	0.00%	0.01%	0.500	0.000
	Brute-force	97.68%	89.31%	2.06%	0.936	0.705
	Botnet	98.92%	0.00%	0.04%	0.500	0.000

Table 5.5: Transfer learning results for extra tree.

Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CSE-CIC-IDS2018 (training)	DoS	100.00%	100.00%	0.00%	1.000	1.000
	Web atk.	99.99%	83.61%	0.00%	0.918	0.899
	Brute-force	100.00%	100.00%	0.00%	1.000	1.000
	Botnet	100.00%	99.99%	0.00%	1.000	1.000
Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CICIDS2017 (test)	DoS	78.55%	43.12%	1.09%	0.710	0.595
	Web atk.	98.48%	0.00%	0.24%	0.499	0.000
	Brute-force	96.56%	0.14%	0.35%	0.499	0.003
	Botnet	98.96%	0.00%	0.02%	0.500	0.000

Table 5.6: Transfer learning results for Gaussian Naive Bayes.

Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CSE-CIC-IDS2018 (training)	DoS	93.07%	97.54%	8.96%	0.943	0.898
	Web atk.	54.18%	100.00%	45.84%	0.771	0.002
	Brute-force	99.72%	100.00%	0.44%	0.998	0.998
	Botnet	93.32%	99.81%	9.11%	0.954	0.890
Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CICIDS2017 (test)	DoS	72.57%	28.30%	2.02%	0.631	0.429
	Web atk.	98.69%	0.00%	100.00%	0.500	0.000
	Brute-force	96.90%	0.00%	100.00%	0.500	0.000
	Botnet	94.78%	34.83%	4.58%	0.651	0.122

Table 5.7: Transfer learning results for KNN.

Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CSE-CIC-IDS2018 (training)	DoS	100.00%	100.00%	0.00%	1.000	1.000
	Web atk.	99.99%	82.40%	2.10%	0.731	0.636
	Brute-force	100.00%	100.00%	0.00%	1.000	1.000
	Botnet	100.00%	100.00%	0.00%	1.000	1.000
Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CICIDS2017 (test)	DoS	79.82%	48.35%	2.10%	0.731	0.636
	Web atk.	98.71%	0.00%	0.01%	0.500	0.000
	Brute-force	95.50%	21.06%	2.12%	0.595	0.225
	Botnet	98.95%	0.34%	0.02%	0.502	0.007

Table 5.8: Transfer learning results for MLP.

Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CSE-CIC-IDS2018 (training)	DoS	100.00%	100.00%	0.00%	1.000	1.000
	Web atk.	99.99%	78.99%	0.00%	0.895	0.862
	Brute-force	100.00%	100.00%	0.00%	1.000	1.000
	Botnet	99.97%	99.91%	0.01%	1.000	0.999
Dataset	Metrics	Accuracy	TPR	FPR	AUROC	F1score
CICIDS2017 (test)	DoS	71.68%	29.84%	4.26%	0.628	0.435
	Web atk.	98.74%	0.00%	100.00%	0.500	0.000
	Brute-force	96.59%	0.01%	0.34%	0.498	0.000
	Botnet	98.97%	0.00%	100.00%	0.500	0.000

The neural network does not achieve good classification scores, contrary to its performance on KDD Cup 99 and NSL-KDD datasets. We deduced that it might be possible to fine-tune the MLP on CICIDS2017 to improve its performance. The neural network would be re-trained on normal traffic corresponding to the network to be monitored (CICIDS2017 in this example) in order to improve its performance. We first want to evaluate the performance of the MLP on CICIDS2017 in classic supervised learning, and then to compare these results with the transfer learning technique.

The MLP has a 75-256-128-64-32-16-1 architecture and uses ReLU as the activation. It is trained on CICIDS2017 on 20 epochs with the Adam optimizer. Figure 5.6 shows the evolution of the AUROC score for each training and for each category of attacks.

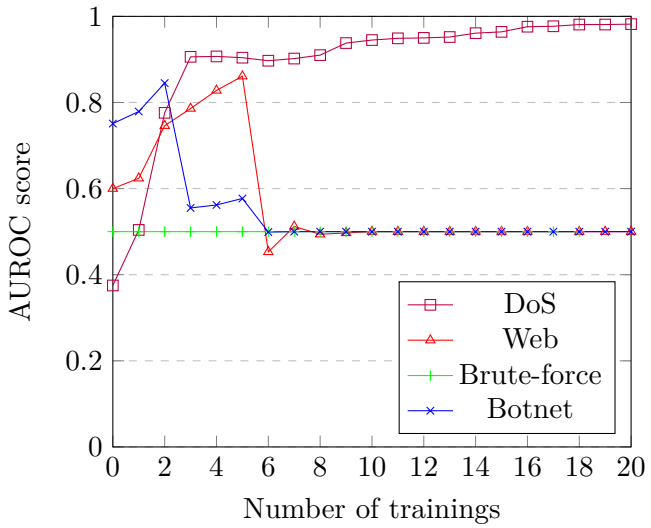

Figure 5.6: AUROC scores for MLP on CICIDS2017

The brute-force attacks are never correctly detected by the neural network, which confirms the results obtained previously with other learning machine models. Web and botnet attacks obtain an AUROC score higher than 0.8, but the quality of their detection deteriorates after 5 and 2 epochs respectively. This phenomenon is probably due to the overfitting caused by the small amount of available training instances. Table 5.9 shows the number of flows labeled as attacks for each category and their ratio within the dataset.

	DoS	Web attacks	Brute-force	Botnet
Number of benign flows	440,031	168,186	432,074	189,067
Number of attacks flows	252,672	2,180	13,835	1,966
Attack ratio	57.42%	1.30%	3.20%	1.04%

Table 5.9: Frequency of attacks within each attack category dataset for CICIDS2017

These ratios correspond to the results obtained in Figure 5.6. The DoS attacks are

very well recognized and have a large number of instances. On the other hand, the three other attacks are poorly detected and have a small number of examples. Ideally, transfer learning should improve the detection of these types of attacks by increasing the amount of data in these categories.

The MLP for transfer learning has the same structure than the previous one. It is first trained on CSE-CIC-IDS2018 on 20 epochs, and then re-trained 20 epochs on CICIDS2017. Figure 5.7 shows the evolution of the AUROC score for each re-training and for each category of attacks.

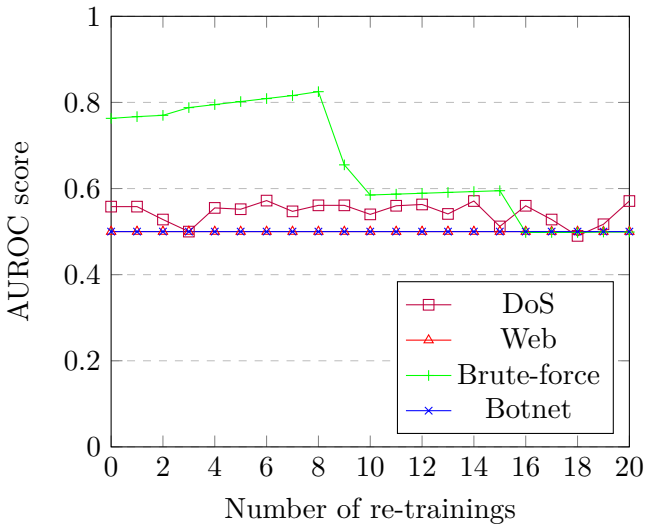

Figure 5.7: AUROC scores for MLP transfer learning with re-training

The results are very different from those shown in Figure 5.6. Web and botnet attacks are correctly detected by the model (AUROC = 0.5). DoS attacks have a slightly higher AUROC score (ranging from 0.5 to 0.58) but still largely insufficient for a reliable detection. Brute-force attacks, which were not detected correctly in classic supervised training, obtain an AUROC higher than 0.8. However, the quality of detection decreases after 8 re-training sessions. This decrease in performance can be explained by the fact that the re-trainings are carried out on benign traffic. This transfer learning technique is a compromise between learning the network to monitor (but without attack) and knowing about the attacks (but on the wrong network). The relative detection performance of each family of attacks can be interpreted using 5.10.

	DoS	Web attacks	Brute-force	Botnet
Number of benign flows	1,442,849	2,096,222	667,626	762,384
Number of attacks flows	654,300	928	380,949	286,191
Attack ratio	45.35%	0.04%	57.06%	37.54%

Table 5.10: Frequency of attacks within each attack category dataset for CIC-IDS-2018

Brute-force attacks are both the best detected and those with the highest attack ratio. DoS and botnet attacks are not correctly detected despite a large amount of attack data. The detection may be too network dependent to be correctly transferred to the target dataset. Finally, the very low number of web attacks (928 samples) explains the misclassification of this category.

This transfer learning technique does not improve the classification of all categories of attack, but achieves good performance for brute-force. Despite the fine-tuning performed on the neural network, the decision tree remains the best model for transfer learning from CSE-CIC-IDS2018 to CICIDS2017. This result can be explained by the low number of data in certain classes, which limits the contribution of more complex algorithms such as neural networks that require more samples. Transfer learning is a promising technique that could achieve very good results with a lot of data and deep neural networks.

Chapters 4 and 5 were dedicated to different supervised training techniques for intrusion detection. It should be noted that all these techniques are therefore based on the learning of a data set, which is contrary to the principle of anomaly detection. It is difficult for this kind of systems to detect attacks never seen before, since they are trained on a database of known attacks. The only attacks they can detect are those that are close enough to other known attacks. Supervised learning on a labeled dataset must therefore be discarded in order to perform real anomaly detection. This method is explored in the next chapter on unsupervised detection.

Chapter 6

Unsupervised Intrusion Detection

This chapter presents an unsupervised anomaly-based intrusion detection solution focused on protocol headers analysis. This approach is tested on a recent and realistic dataset (CICIDS2017) over a 4-day period. Each protocol is converted to a set of normalized numeric features, which are processed by 5 neural network architectures. The output of these algorithms is an anomaly score, which is normalized and combined with the anomaly scores of other protocols.

6.1 Protocol-based intrusion detection

6.1.1 Introduction

Anomaly detection is most often based on machine learning algorithms with supervised learning. This is a problem as anomaly detection specifically learns the behavior of a given network: it is therefore not directly transferable to another computer network. The IDS must be re-trained on the network where it is deployed with a dataset containing labelled attack data. Unfortunately, it is difficult in practice to set up such a dataset, which requires a dedicated traffic generator [95]. Furthermore, such an IDS is mainly trained to detect attacks that are already known, which limits its value compared to a signature-based IDS.

On the other hand, unsupervised learning does not require any attacks in its training data. It is therefore much easier to deploy in reality and not biased by a selection of known attacks [96]. However, unsupervised learning has lower detection rates compared to supervised learning. It is easier to train a machine leaning model to recognize a pattern than to detect an anomalous element in a dataset. The advantages of this type of training therefore come at the cost of the accuracy of the IDS.

Anomaly detection raises a semantic problem on the very definition of an intrusion. In chapter 2, we defined an intrusion as all unauthorized activities that compromise the confidentiality, integrity, or availability of an information system. This definition, however, is too general in the context of intrusion detection on a computer network. Network-based IDSs cannot detect an attack on the hardware for instance, since they

are not provided with the relevant information for this kind of intrusion.

Indeed, the input parameters fully define the detection spectrum of an IDS. These parameters must therefore be chosen according to the attacks to be detected. A distinction must be made between network attacks (e.g. denial of service) and attacks that pass through the network infrastructure (e.g. malware). While network attacks must be detected by a network-based IDS, attacks that pass through the network are difficult to identify because their main characteristics are not captured (e.g. system calls) in the input parameters. Ideally, they should be detected by a host-based IDS.

However, this distinction is rarely well defined. Certain types of attacks such as brute-force can be detected by both a network-based IDS (many quasi-identical packets) or a host-based IDS (many unsuccessful login attempts). Figure 6.1 shows that any attack has a different ratio between its observable effect on the target system or on the network. Most attacks impact both, although some are specific to a single dimension. For instance, ARP spoofing involves an attacker sending spoofed ARP messages on a network. The attacker's MAC address is associated with the IP address of another host, meaning that any traffic destined for that IP address is sent to the attacker instead. It has no effect on targeted systems and therefore cannot be detected by a host-based IDS.

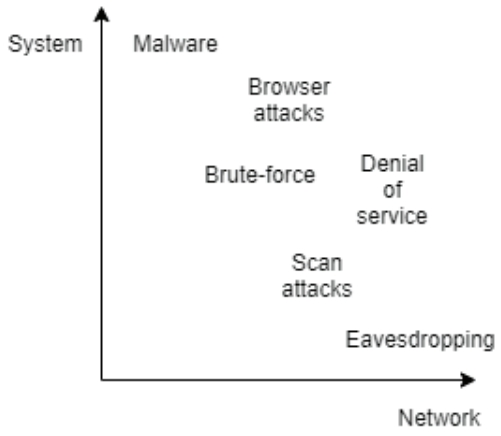

Figure 6.1: Graph of attacks according to their effects on the network or system.

It is therefore difficult to restrict the number of attacks that a network-based IDS must be able to detect. We consider that our IDS must be able to detect all types of attacks, knowing that it is more difficult to identify some of them. It is important to feed it with as many relevant parameters as possible to detect a wide variety of attacks.

6.1.2 Data processing

Network traffic analysis is mainly divided into flow and packet analysis. RFC 6437 defines a flow as "a sequence of packets sent from a particular source to a particular unicast, anycast, or multicast destination that a node desires to label as a flow" [97]. The transition from packets to flows leads to a loss of the information contained in the

packets. The lost information might not necessarily be useful for intrusion detection, and this compression speeds up processing compared to packet analysis.

However, network protocols are a form of artificial language, while flows are only an aggregation of them. Recent work in Natural Language Processing (NLP) has shown excellent results in language modeling with machine learning algorithms [98]. Although network protocols have many differences with natural languages, their low variance makes them easier to predict than flows. Modeling the language of the protocols would make it possible to recognize headers belonging to this language. Headers not belonging to the language can therefore be considered as anomalies. For these reasons, we chose to train machine learning algorithms to model the behavior of protocol headers.

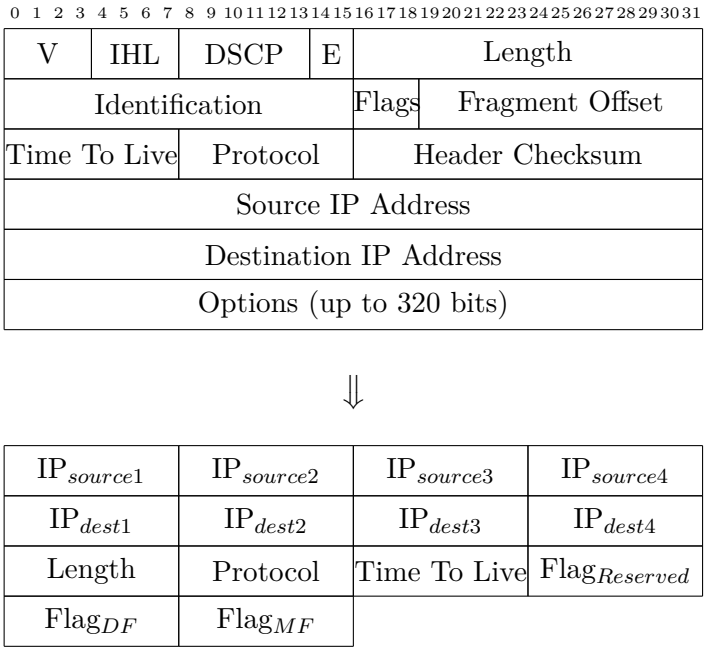

Figure 6.2: IPv4 Header Feature Extraction.

Received packets are decapsulated and the protocols are processed independently. For each protocol, the relevant features are extracted as shown in Figure 6.2. These features are automatically normalized between 0 and 1 using the limit values given in their documentation. The standardised and error-cleaned data are then stored in a database.

6.1.3 Framework

Fig. 6.3 illustrates the ensemble learning process that is used to infer a packet anomaly from the analysis of multiple protocols. First, features are extracted from protocol headers as described previously. These features are analyzed by a machine learning model to output an anomaly score. Finally, these anomaly scores are normalized and

combined to obtain an anomaly score for the entire packet.

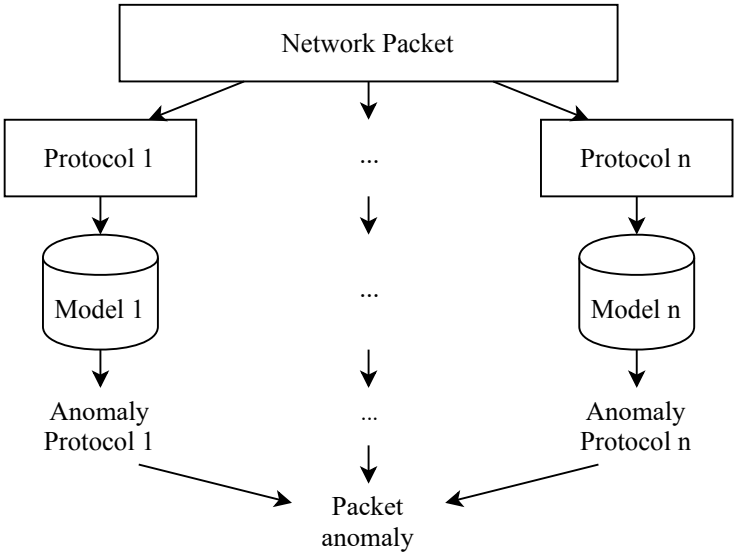

Figure 6.3: Protocol-based Ensemble Learning.

The purpose of protocol-based ensemble learning is to detect attacks related to one protocol (e.g., SYN flooding) and to increase confidence in the predictions when attacks are detected on multiple protocols. Moreover, it also allows to dynamically add or remove protocols according to the monitored network. The anomaly score reflects the fact that the analyzed protocol header is unexpected by the machine learning model. Different combination rules can be used to obtain the packet's anomaly score. For example, it is possible to keep only the highest anomaly score among the headers, but this method produces many false positives. On the other hand, keeping only the lowest anomaly score may decrease the attack detection rate. We chose to average the anomaly scores of the different protocol headers. This method could be weighted to better represent the importance of each protocol in attack detection. However, in an anomaly detection process, these weights would have been determined by already known attacks. It could therefore have biased the model towards a poorer recognition of unknown attacks.

All packets detected as anomalies cannot be considered as attacks. A second processing step is therefore necessary to analyze the relevance of these anomalies. We make the hypothesis that a computer attack is composed of a succession of anomalies. The goal of the IDS is not to detect all the packets that constitute an attack, but to detect the attack itself. Packet anomalies must therefore be combined to erase isolated events and highlight nearby anomalies. We choose a moving average as a combination rule, using only the packet anomalies in the past to be able to do real-time intrusion detection. The IDS considers an attack is ongoing when this final anomaly score exceeds a predefined threshold.

Algorithm 2: Packet Anomaly Prediction

```
input  : Network packet
output: Packet anomaly (number between 0 and 1)
    extract protocol headers from network packet;
    foreach protocol header do
        select list of features;
        convert categorical features into numerical features;
        normalize features between 0 and 1;
        if protocol header has a trained neural network then
            predict anomaly score;
        else
            train neural network;
    average anomaly scores;
```

6.1.4 The problem with metrics

Anomaly-based intrusion detection usually relies on machine learning. Different metrics are used to measure the performance of these machine learning algorithms: TPR, FPR, F1-score, AUROC, etc. [99] The very choice of these metrics is problematic, as none of them can perfectly represent the quality of the algorithm's detection alone. For instance, F1-score completely ignores true negatives. However, the main problem is that these metrics do not reflect the problem that the algorithm is trying to solve.

Correctly classifying flows or packets and detecting attacks are two distinct problems. Nevertheless, the metrics inherited from machine learning are only relevant to the first one. Indeed, intrusions are most often composed of several flows or packets [100]. However, detecting all the flows or packets of an unknown attacks is unrealistic without producing a lot of false positives. Fortunately, anomaly scores can be correlated to better estimate the probability of a given sequence being an attack. Therefore, detecting a small portion of the flows or packets of an attack is enough to deduce that an intrusion is ongoing. This is why metrics in intrusion detection should measure the performance of the attack classification.

From the perspective of network administrators, we identified 3 expected outputs:

1. Correct alerts. An IDS must be able to detect attacks with a low false positive rate. TPR, FPR, F1-score, AUROC, etc. are relevant metrics to measure the performance of attack classification.

2. Low-latency alerts. Intrusion detection is often followed by an intrusion reaction stage. This reaction can be manual or automatic with an IPS. Reactions are all the more effective if the attack is detected early enough. This is why we propose a new metric: the time between the beginning of the attack and its detection (called Time Before Detection).

3. Information gathering. Collecting as much information as possible about the attack is also a very important feature for an IDS. This information can then be used to stop or to block the ongoing attack. However, this thesis does not cover intrusion reaction, so we will not collect information on detected attacks to illustrate this point.

6.2 Ensemble Learning

6.2.1 Introduction

In this section, we test 5 different architectures of neural networks trained in a unsupervised way on protocol headers. The best models are then combined to obtain the most accurate detection possible. Finally, a post-processing step refines the results and eliminates most false alarms. We argue that the metrics usually used to evaluate the quality of an IDS (TPR, FPR, F1-score...) are not appropriate to effectively measure its performance in a real situation. A new metric is proposed, focusing on the detection speed of attacks, and not on a correct classification rate. This metric is defined as the time between the beginning of the attack and its actual detection.

Network intrusion detection has been extensively studied in the past decades. Clustering is one of the most popular unsupervised method to find anomalies in a dataset. Leung and Leckie [101] show an application of this technique with a clustering algorithm on KDD Cup 99, specifically designed for intrusion detection.

More recently, autoencoders and deep autoencoders have been used for their ability to reconstruct their input. However, like other methods, they are prone to generate many false positives. Kotani and Sekiya [102] developed a flow-based method with robust autoencoders to reduce the false positive rate on a real-world traffic dataset. Mirza and Cosan [103] tested different RNN units for autoencoders to find the best architecture for packet payload reconstruction. This technique is complementary to ours since we do not analyze protocol payload data in this study.

6.2.2 Dataset

This work requires data that is representative of a real network, including both malicious and normal traffic. This dataset needs to be labelled in order to measure the performance of the IDS. We chose the CICIDS2017 dataset for its realistic network and credible attacks [104].

CICIDS2017 consists of 3 119 345 labeled network flows (83 features) and 56 329 679 network packets. This dataset is also highly imbalanced, with 83.34% normal flows against 0.00039% flows labelled as “Heartbleed” [26]. In a supervised learning task, data augmentation would be a good solution to rebalance the 15 classes of CICIDS2017. However, in our unsupervised learning task, data augmentation cannot be used: Heartbleed attacks will simply be more difficult to detect compared to more prevalent classes.

To the best of our knowledge, we found an undocumented shortcoming of CICIDS2017. The attacker’s private IP address (172.16.0.1) remains the same for most

<table>
<tr><th></th><th>Monday</th><th>Tuesday</th><th>Wednesday</th><th>Thursday</th><th>Friday</th></tr>
<tr><td>9:00</td><td rowspan="8">Benign traffic</td><td>FTP-Patator</td><td>DoS Slowloris</td><td>Brute-force</td><td>Benign traffic</td></tr>
<tr><td>10:00</td><td rowspan="3">Benign traffic</td><td>Slowhttptest Hulk</td><td>Injections</td><td>Botnet ARES</td></tr>
<tr><td>11:00</td><td>DoS GoldenEye</td><td rowspan="2">Benign traffic</td><td rowspan="2">Benign traffic</td></tr>
<tr><td>12:00</td><td rowspan="3">Benign traffic</td></tr>
<tr><td>13:00</td><td rowspan="3">Port scan</td></tr>
<tr><td>14:00</td><td>SSH-Patator</td><td rowspan="2">Infiltration Dropbox</td></tr>
<tr><td>15:00</td><td>Benign traffic</td><td>Heartbleed</td></tr>
<tr><td>16:00</td><td>Benign traffic</td><td>Benign traffic</td><td>Benign traffic</td><td>DDoS LOIT</td></tr>
</table>

Table 6.1: Attack schedule of CICIDS2017.

attacks (Tuesday, Wednesday, Thursday morning and Friday afternoon), contrary to the authors' indications. Furthermore, this IP address is used almost exclusively for intrusions, which makes it very easy to detect. We could not identify other attacking IP addresses from the network packets. We ensured that the neural networks used in this study did not simply learn to detect the IP address 172.16.0.1. However, this feature was considered too important to be simply removed from the dataset.

In this work, we focus on the 8 protocols most represented in CICIDS2017: Frame, Ethernet, ARP, IP, TCP, UDP, DNS, and HTTP. Other protocols could be added if necessary. First, these protocols headers need to be extracted from the packets contained in the dataset. Then, a set of features is selected from the variables contained in these headers. Categorical features are converted into numerical features by one-hot encoding or label encoding. All these features are then normalized between 0 and 1 (min-max scaler).

Finally, each packet must be labelled as an attack or a benign packet. Note that only flows are labelled by the authors of CICIDS2017. Fortunately, they detailed their labelling process in the original paper [104] with IP addresses and timestamps. Although these details are probably not exhaustive enough to properly label each packet during an intrusion, they are sufficient to detect continous attacks. This preprocessing framework creates 8 datasets for each day that are stored in SQL databases.

This process has been applied on Monday (11 701 690 packets – 11GB), Tuesday (11 543 109 packets – 11GB), Wednesday (13 781 563 packets – 13GB), and Thursday (9 314 199 packets – 7.8GB). Friday (9 989 118 packets – 8.3GB) has been excluded to reduce computing time and because of the difficulty of labelling numerous very short attacks.

6.2.3 Neural network architectures

The objective of the neural networks is to learn the behavior of each protocol header. Different neural network architectures can be used to solve this problem. Five architectures are studied: deep autoencoders, deep MLPs, LSTMs, Bidirectional LSTMs (BiLSTMs), and Generative Adversarial Networks (GANs). Each architecture and its specific use is

briefly described in the following.

- Autoencoder is a feedforward neural network with a bottleneck to force the network to learn a compressed representation of the original input. We use a wide topology with a $len(input)$-256-128-64-32-64-128-256-$len(input)$ structure and ReLU as the activation function. Autoencoders learn to minimize the distance between the output $\overline{x}$ and the input x. The Mean Squared Error (MSE) is employed to evaluate performance during training, and as an anomaly score during test. This approach assumes that an attack will be more difficult to reconstruct for the network than a normal protocol header.

- MLP is another feedforward neural network, with a $len(input)$-512-256-128-64-$len(input)$ structure and ReLU as the activation function. This MLP receives the 50 previous protocol headers as input and tries to predict the next protocol header. This input window provides context to the network compared to the previous approach. The distances between each prediction and the actual next protocol header are used to evaluate performance during training, and as an anomaly score during test.

- LSTM is a recurrent neural network that can learn long-term dependencies. Its input is similar to the previous architecture, with a window of 40 protocol headers. Its purpose is not to rebuild part of the input, but to predict the next protocol header. On the other hand, BiLSTMs run input data once from beginning to the end, and once from end to beginning to understand context better. Their input is comprised of the 20 previous protocol headers and the 20 next protocol headers. BiLSTMs try to predict the protocol header in the middle of this input window. Both architectures have $len(input)$ input units, 400 LSTM or BiLSTM units, $2 \times len(input)$ ReLU units, and $len(input)$ sigmoid units.

- GAN is a deep neural network architecture composed of two networks: a generator (1000-128-256-512-$len(input)$ with LeakyReLU and dropout) and a discriminator ($len(input)$-256-128-64-1 with LeakyReLU and dropout). The first net generates new data, while the other classifies each example as real (i.e., from the training set) or fake (i.e., generated). GANs are trained on Monday with benign traffic. Only discriminators are then used as classifiers for the next days. This approach assumes that an attack will not be recognized by the discriminator as part of the training set.

6.2.4 Experiments

The proposed framework was implemented on two GTX 1080 Ti GPUs and an Intel i5-7500 CPU with 64GB of RAM. 8 models of each neural network architecture were trained on Monday's traffic on each protocol: Frame (11 701 690 instances), Ethernet (11 701 690 instances), ARP (46 971 instances), IP (11 618 823 instances), TCP

(10 710 204 instances), UDP (934 997 instances), DNS (788 288 instances), and HTTP (107 681 instances).

Training time on the frame protocol depends on the architecture, but takes on average between 9 and 15 hours (with early stopping and cyclical learning rate). Autoencoders are the fastest architecture to train with 10 minutes 13 seconds per epoch on the frame protocol. BiLSTMs are the slowest architecture to train with 16 minutes 53 seconds per epoch on the same protocol.

Models are tested on Tuesday, Wednesday, and Thursday. Attacks can be deduced from packet anomalies, but these final scores are very noisy. A moving average with a window of 10 000 packets is applied to erase isolated anomalies and to aggregate groups of anomalies into continuous attacks. Since this detection framework is intended for real world application, the moving average is only applied on past packets. Anomalies are normalized between 0 and 1, and a threshold of 0.8 is chosen to categorize attack packets.

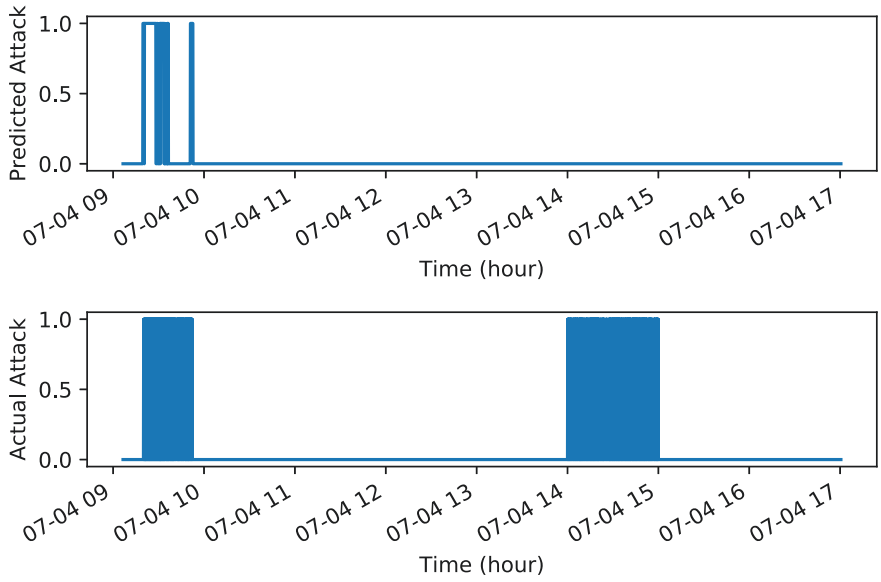

Figure 6.4: Autoencoders Predictions of Attacks on Tuesday.

Results show that GANs produce too many false positives to be used. Autoencoder is the only architecture capable of detecting the FTP-Patator attack on Tuesday morning (see Fig. 6.4). BiLSTM outperforms other architectures for detecting Wednesday and Thursday attacks (see Fig. 6.5 and Fig. 6.6). These results are sufficiently uncorrelated to motivate a new step of ensemble learning, by averaging autoencoders and BiLSTMs packet anomalies. This final model successfully detects 7 out of 11 attacks not seen during the training phase, without any false positives. Table 6.2 summarizes the Times Before Detection for each attack.

The infiltration attack (Thursday afternoon) is the only multi-step attack, comprised

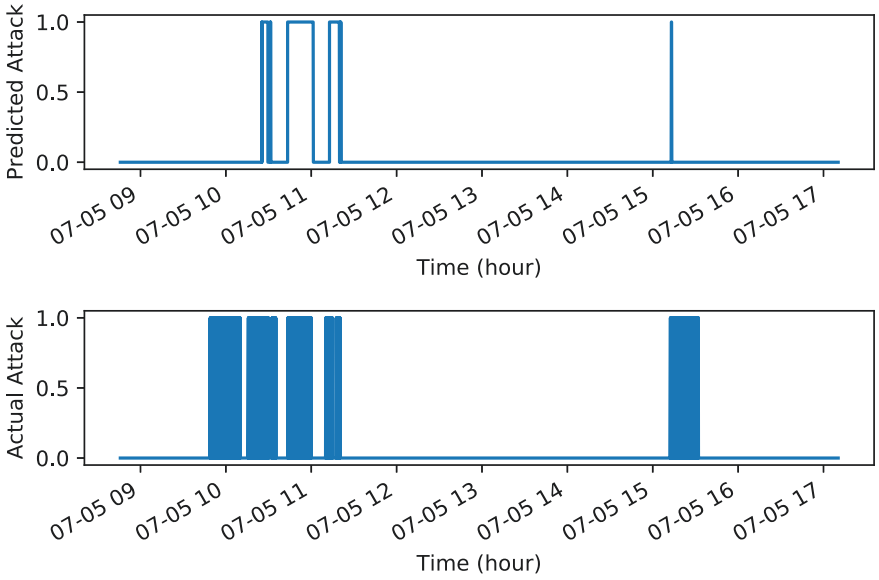

Figure 6.5: BiLSTMs Predictions of Attacks on Wednesday.

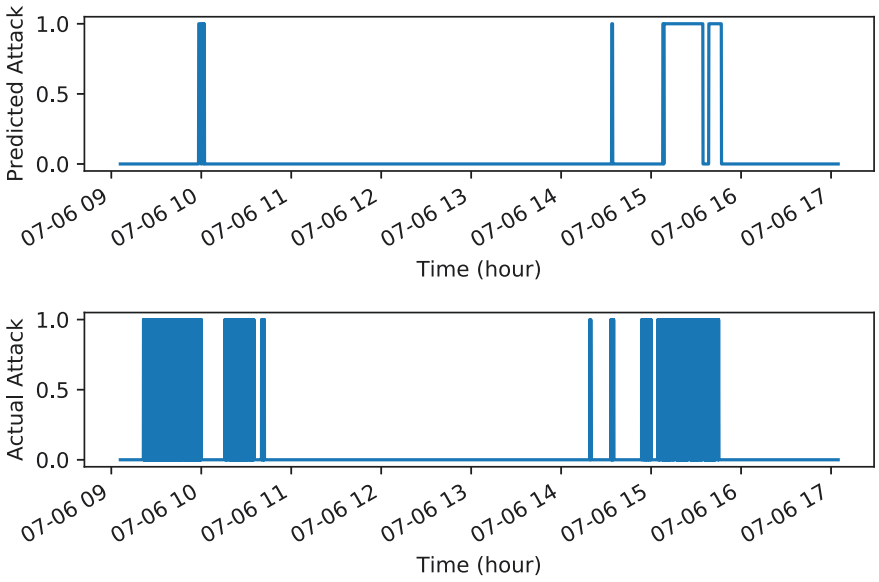

Figure 6.6: BiLSTMs Predictions of Attacks on Thursday.

Table 6.2: Time Before Detection for CICIDS2017 Attacks

	Time Before Detection (s)
FTP-Patator	28.72
SSH-Patator	Not detected
DoS slowloris	Not detected
DoS Slowhttptest	550.10
DoS Hulk	23.60
DoS GoldenEye	170.07
Heartbleed	175.78
Web Brute-force	2293.03
XSS	Not detected
SQL injection	Not detected
Infiltration	874.92

of 3 stages: Meta exploit Win Vista (14:19, 14:20-14:21, and 14:33-14:35), Infiltration – Cool disk – MAC (14:53-15:00) and Infiltration – Dropbox download Win Vista (15:04-15:45). The intrusion is first detected between 14:33 and 14:35, and then between 15:10 and 15:31. Five attacks are detected in less than 5 minutes, allowing for a quick response. Some attacks are more difficult to identify because of a slowly increasing anomaly score (SSH-Patator, DoS slowloris, Web Brute-force, Infiltration). Finally, XSS attacks and SQL injections have given signals that are too weak to be detected by the IDS.

6.2.5 Conclusions

Our proposed detection framework offers a high detection rate while solving the main problem of anomaly detection (high FPR). Its deployment in a real network does not require a traffic generator or transfer learning like supervised learning methods would. The protocol-based detection is highly customizable, with the ability to add or remove protocols depending on the monitored network. The main drawback of this method is a high training time (up to 14 hours for a single protocol) compared to other unsupervised techniques. Feature selection and a reduced input window for BiLSTMs could help speed up the training.

Attack recognition is also an important information for intrusion reaction. Indeed, for example, mitigation methods for DDoS attacks and targeted infiltrations are very different [105]. A multi-class classifier could analyze protocol anomalies to identify patterns specific to certain attacks. This classifier would be independent of the network and could be reused in all implementations. This system could not precisely classify unknown attacks but output a family of attacks (DoS, probe, etc.).

Chapter 7

Predicting Bandwidth Utilization

Predicting the bandwidth utilization on network links can be extremely useful for detecting congestion in order to correct it before it occurs. In this chapter, we present a solution to predict the bandwidth utilization between different network links with a very high accuracy. Section 1 introduces the concepts of bandwidth utilization prediction. Section 2 discusses related work about congestion detection. Section 3 introduces the data generation process using a simulated network. Section 4 describes the preprocessing stage, from data collection to the creation of the dataset. Section 5 presents the machine learning models used to predict the bandwidth utilization on network links and our experimental results. Finally, section 6 discusses areas of future work and concludes this chapter.

7.1 Introduction

All companies that offer network services (ISPs, server hosting services etc.) use mechanisms to monitor link utilization. These mechanisms usually involve network interface monitoring and collection of performance statistics (e.g. with SNMP [106]), monitoring of flows (e.g. with NetFlow [107], sFlow [108], etc.) or capturing packets and further analyzing them with a specific tool [109]. Detection of high network utilization is a problem that needs to be addressed efficiently since it usually causes packet loss, increased latency due to buffering of packets, and interference with TCP's congestion-avoidance algorithms. The result is a degradation of the Quality of Service (QoS) of the network.

There are numerous ways to circumvent the problem like allocating more bandwidth to accommodate for the increased traffic, prioritizing important traffic through QoS, blocking undesirable traffic, or load balancing the traffic across multiple paths [110]. However, all of these solutions come after the bandwidth problem has been detected. Predicting the future bandwidth consumption would allow to proactively correct this problem.

One of the most important metrics for network performance evaluation is the network link utilization at any given time. Link utilization is usually expressed as a percentage of the total capacity of a network link. For example an 100 Mbps Ethernet link that

carries 20 Mbps of traffic exhibits 20% utilization. The higher the percentage of usage, the lower the quality of the link, resulting in packet loss and increased latency.

Our solution predicts network utilization using machine learning algorithms. A simulated network is created to collect data from network and system resource statistics. They are processed with feature engineering and merged to create a dataset, which is fed to the machine learning algorithms. Three models are tested: ARIMA, MLP and LSTM. Our goal is to identify weak signals among the features of the dataset to predict the bandwidth utilization.

7.2 Related Work

Congestion detection is a useful tool to improve the performance of any type of network. Many solutions have been proposed at the network protocol level or at the application level. The use of machine learning algorithms has gained prominence over the last two decades in the literature.

Devare and Kumar [111] applied time series analysis to congestion control using clustering and classification techniques. They generated traffic patterns for a number of clients, selected relevant data and preprocessed them to build a database. Different machine learning classification algorithms are trained with these data and used to predict the future traffic intensity. Countermeasures are then applied to improve the performance of the network.

Singhal and Yadav [112] used neural networks to detect congestion in wireless sensor networks. The authors created their own dataset using NS-2 [113] to generate a random traffic. Three features are used as inputs for the machine learning algorithm: the number of participants, the traffic rate, and the buffer occupancy. This neural network has a 3-10-10-1 structure and predicts a level of congestion (low, medium, or high).

Madalgi and Kumar [114] explored a similar idea with two congestion detection classifiers. Their dataset is also generated with NS-2 and machine learning algorithms are used to predict the same three levels of congestion. The authors compared the performance of a MLP and a decision tree (M5 model tree). They show that the decision tree is trained faster (0.53 second against 8.25 seconds) and outperforms the MLP in terms of accuracy, true positive rate, and false positive rate.

The same authors [115] proposed another work using Support Vector Machines (SVMs). They specifically used LibSVM and Sequential Minimal Optimization (SMO), with Radial Basis Function (RBF) as kernel. Different parameters were tweaked to find the best values to improve the classification accuracy. The best model is trained in 18.81 seconds and achieves slightly lower results than the previous M5 decision tree.

Zhang et al. [116] worked on a specific type of network congestion: low-rate denial of service attacks. This type of attacks exploits the TCP congestion-control mechanism to deplete the resources of the target. The authors used the Power Spectral Density (PSD) entropy function to reduce the number of calculations: a flow is classified as normal below a certain threshold, and as an attack above a second threshold. A SVM classifies uncertain connections between these two thresholds, using 8 features. The experimental

results show that this solution can detect 99.19% of the low-rate denial of service attacks in the dataset.

7.3 Data Generation Using A Simulated Network

A testbed network was created in order to collect data for the machine learning algorithms. Regular TCP and UDP traffic was generated which occasionally caused heavily loaded links in order to simulate real-world scenarios. The network topology is illustrated in Figure 7.1.

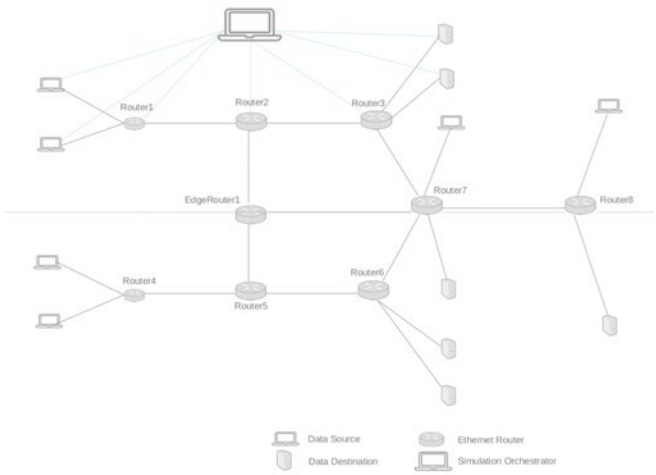

Figure 7.1: Topography of the Simulated Network.

Virtualbox was used to simulate the network with each end host and router being a separate virtual machine. Network links were configured with a maximum capacity of 100 Mbps and with 2 ms of latency. Finally, a separate VM called Orchestrator was created to orchestrate the simulation phase on the network.

iPerf3 was used to simulate traffic in the network [117]. iPerf3 is primarily used for bandwidth measurements of network links by generating TCP and UDP flows with certain parameters. Using iPerf3 as a traffic generator has its limitations as it cannot always generate realistic network traffic. For example, it mainly generates packets with fixed inter-departure times and cannot create bursty traffic. It is, however, easily scriptable and allowed us to quickly generate training data for our models. Moreover, we have randomized some of the flow parameters to introduce some diversity into the network traffic. The random flow parameters are summarized in Table 7.1.

The bitrate, flow duration, and flow inter-arrival time intervals were chosen to ensure that maximum capacity would be reached on certain or all links, even for a short period of time. In other words, we had to make sure that we had data ranging from zero traffic to maximum capacity while keeping simulation times relatively short.

During the simulation phase, a script running on the Orchestrator was in charge

Table 7.1: Randomized Flow Parameters

Parameter	Values	Distribution
Protocol	TCP, UDP	Uniform
Bitrate	Average: 2 Mbps Std: 500Kbps	Normal
Flow Duration	[30,500] seconds	Uniform
Packet Length	[200,1472] bytes	Uniform
Flow Inter-arrival Time	[20,2000] seconds	Uniform
Source/Destination couple	IP of the end hosts	Uniform
iPerf3 Server Port	[5001,5500]	Uniform

of starting the data collection mechanism on each one of the routers. It started every $[20, 2000]$ seconds an iPerf3 server on a destination machine and an iPerf3 client on a source machine, thus creating a new data flow with certain characteristics. The results presented later in this paper come from a three-day simulation, where all the interfaces of each one of the routers were monitored. The data were collected in the form of text files with a predetermined frequency (every 3 seconds). The simulation generated around 85 000 files per interface for a total of 2 million files (12 GB) for all the 22 monitored interfaces.

7.4 Preprocessing Stage

7.4.1 Data Collection

Machine learning algorithms for supervised learning require labelled data as input. The prediction quality is directly related to the relevance of these data. Two tools were used in order to obtain a maximum of information on the bandwidth of the network links: Netmate and Dstat.

Netmate (Network Measuring and Accounting Meter) [118] is a network measurement tool that can collect a number of network statistics, such as packet volumes and sizes, packet inter-arrival times, and flow duration. There are 46 features collected by Netmate for every active flow at the time of the capture. A separate instance of Netmate was run for every interface on a router. This tool was configured to export its output as a .csv file every 3 seconds for every interface. Every .csv file was timestamped with the seconds since epoch format.

Dstat [119] is a versatile Linux tool for generating system resource statistics from numerous system components like CPU, RAM, input/output devices, networks connections and interfaces, and others. This tool is highly configurable and can output the collected data into a .csv file for further processing. Dstat collected 29 statistics with our configuration.

7.4.2 Feature Engineering

Data from Netmate and Dstat require a preprocessing phase in order to create datasets in a format suitable for machine learning algorithms. Given the fact that Netmate generates its statistics for every active flow, a feature engineering procedure is necessary to create new features that would describe the status of the link collectively. Mathematical functions like $min()$, $max()$, $mean()$, $sum()$ and $count()$ were applied to the Netmate data to obtain 86 new features. For instance, the total number of packets can be calculated with the sum of the number of packets for each of the active flows: $total\_pkt\_vol = sum(pkt\_volume\_of\_each\_flow)$. These 86 features are then concatenated with the 29 features coming from Dstat using the timestamp value to synchronize the output of the two tools. After this feature engineering step, 116 features are collected from every interface of each router in our network every 3 seconds.

The feature engineering process is illustrated in Figure 7.2.

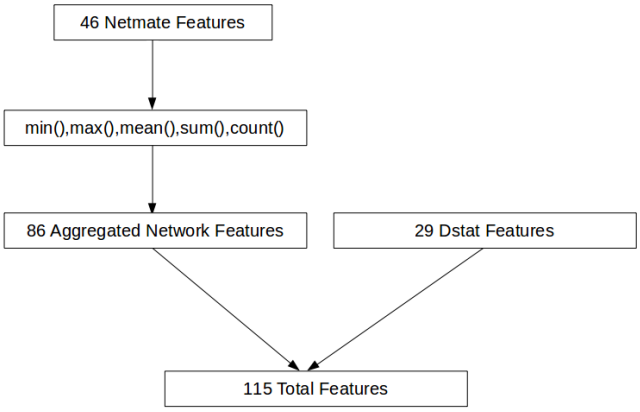

Figure 7.2: Feature Engineering Workflow.

7.4.3 Creation of a Dataset

A dataset for supervised learning needs to be labeled for the training process. The two most important features to predict the link usage are the *download_bitrate* and *upload_bitrate* provided by Dstat. A new feature called *max_bitrate* is created by choosing the maximum of the two values and dividing it by the nominal maximum capacity of the link (100 Mbps). In other words, the new column gives the link usage ratio, either in the uplink or the downlink direction. The *max_bitrate* column is then duplicated to create the $future\_bitrate$ column, which is shifted forward by an offset value. Machine learning algorithms are trained to predict this $future\_bitrate$ feature, namely the future *max_bitrate* of the link.

The offset value along with the frequency of the data collection give the time depth of the future predictions. Different values were tested between 1 and 40. The quality of the prediction decreases significantly for an offset greater than 20. The value of 5 constitutes

an optimal comprise between time and accuracy, without being too resource-intensive or time consuming. A frequency of 3 seconds with an offset of 5 means that our model will predict the $future\_bitrate$ value 15 seconds in the future. 15 seconds is enough time for a reaction mechanism to correct future congestion. Finally, the dataset is normalized between 0 and 1 using a min-max scaler. This normalization step is important for the training process, since the difference in the features scales can cause problems during the training.

7.5 Experiments

7.5.1 Machine learning algorithms

Three types of machine learning algorithms are tested and evaluated in this paper: the Autoregressive Integrated Moving Average (ARIMA), the Multilayer Perceptron (MLP) and the Long Short-Term Memory network (LSTM). These algorithms are trained in a supervised way and try to predict the future bandwidth utilization based on 1 feature for ARIMA ($future\_bitrate$), and 115 features for the two neural networks. The dataset was randomly divided into k subsets of equal size. The first subset was used for validation and the models were trained on k-1 subsets (k-fold cross-validation). This technique gives a more accurate estimate of the models' performance by reducing bias and sample variability between training and test data [120].

ARIMA is a widely used approach for analyzing and forecasting time series data. If necessary, time series are made stationary by differencing or applying nonlinear transformations. A stationary time series has the property that the mean, variance, and autocorrelation are constant over time. This type of data is easier to predict since its statistical properties will be the same in the future. The ARIMA model is fitted on a single feature: the future bandwidth utilization $future\_bitrate$.

MLPs are a classical feedforward neural network architecture. For this work, a 116-256-128-64-1 topology is used, with ReLU as the activation function. MLPs are fast to train and suitable for regression prediction problems, particularly with tabular datasets. Hyperparameters were tuned manually, with a batch size of 128 during 10 epochs, using the Adam optimizer.

LSTMs are a special kind of recurrent neural networks, capable of learning long-term dependencies [121]. They are designed to recognize patterns in sequence of data, with an additional temporal dimension. This feature is particularly relevant to our problem, which requires learning new network behaviors while remembering past events. Hyperparameters were tuned manually, for a batch size of 128 during 10 epochs, using the Adam optimizer. The final architecture consists of 300 LSTM units and 116 ReLu units.

These three models were chosen to assess the complexity of the problem being modelled. ARIMA is able to model periodic phenomena such as seasonal sales with good accuracy. If it proves to be effective, this will mean that the prediction of network link usage can be summarized in one feature. MLP and LSTM are two models that use

all 115 features for their predictions. The difference is that LSTMs keep a memory of past events, which can be useful for predicting future bandwidth utilization. This type of neural network is typically used in time series problems. But an MLP processes data faster, which is an advantage for real-time use. This architecture will therefore be preferred to the LSTM if the results are similar.

7.5.2 Model Validation and Results

After the dataframe creation phase, one dataset is created for each of the 22 monitored interfaces. Models are validated using k-fold cross-validation: they are trained on k-1 datasets and validated on the remaining one. By shuffling the different folds, we can thus obtain all the permutations of the k-1 datasets and obtain k evaluation scores for each model. This technique also ensures that the model is validated on an interface never seen before, which is a realistic use case.

Due to memory limitations and the fact that some interfaces were reciprocal (in the sense that they are directly connected to each other and the upstream traffic of one is the downstream traffic of the other), we cherry picked 8 interfaces for the evaluation. Four metrics commonly used in time series forecasting were chosen to evaluate the performance of the models: bias (systematic deviation from the actual values), Mean Absolute Error (MAE), Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). Table 7.2 shows the averaged values of these metrics for each model.

Table 7.2: Averaged k-fold cross-validation scores

Model	Avg. Bias	Avg. MAE	Avg. MSE	Avg. RMSE
ARIMA	0.161129	0.162010	0.071085	0.266617
MLP	0.001604	0.022826	0.002965	0.051939
LSTM	-0.002142	0.004272	0.001444	0.012233

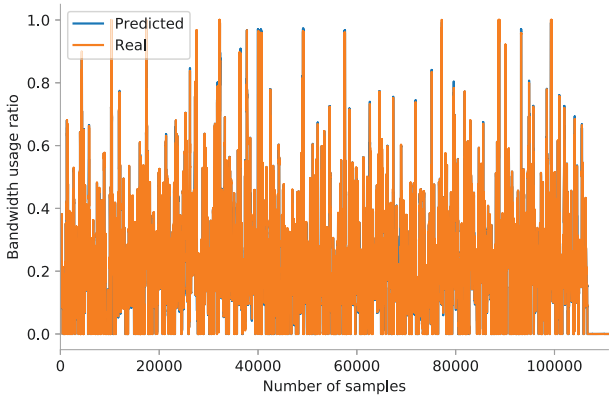

Figure 7.3: LSTM predictions vs. actual values for one interface.

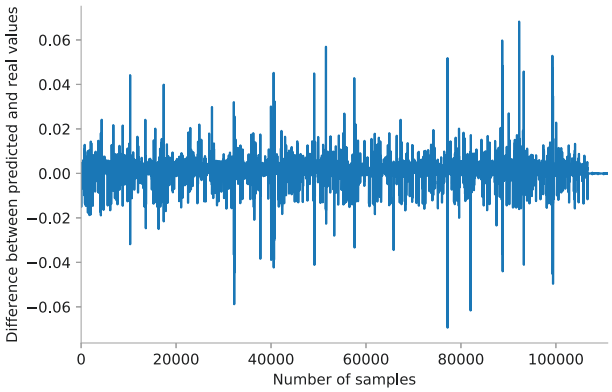

Figure 7.4: LSTM difference between predicted and actual values for one interface.

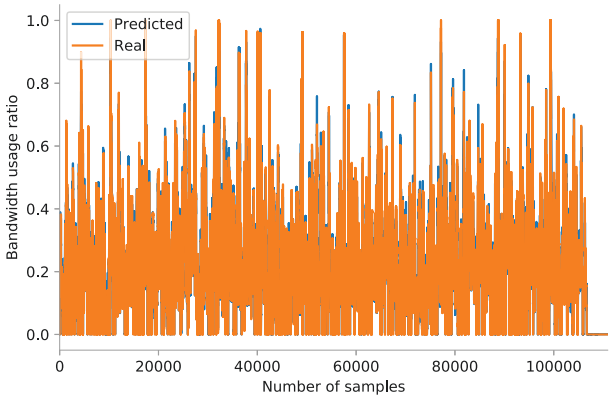

Figure 7.5: MLP predictions vs. actual values for one interface.

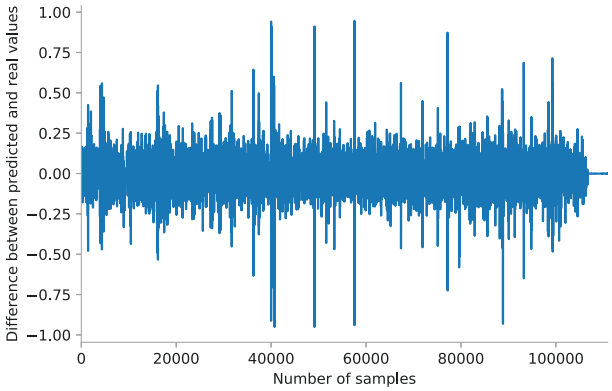

Figure 7.6: MLP difference between predicted and actual values for one interface.

It is immediately apparent from the results that ARIMA's predictions have a high bias and a poor accuracy compared to the other models. The errors produced by ARIMA are too significant for this model to be used in a real context. These poor results can be attributed to the fact that the model is fitted on a single feature. The other features could contribute greatly to the quality of the algorithm's prediction.

LSTM models perform better than MLPs. For all interfaces the difference between the actual value and the predicted one rarely exceeded 3%, which means that the future bandwidth consumption is predicted with an error of +/- 300 Kbps on 100 Mbps links. On the other hand, the difference for the MLP sometimes exceeded 20%, which might cause significant underestimation or overestimation of the future link capacity. The ability of LSTM models to recall past variations in bandwidth usage may explain the high quality of their predictions.

In addition, the results for some interfaces were significantly better than for others. The results were worse when the interface had higher variance of the traffic carried during the simulation phase. In other words, the predictions were much closer to the real values for interfaces that were less utilized during simulation than for interfaces that had seen the whole spectrum from 0 to 100 Mbps.

7.5.3 Real-Time Prediction

After having trained and evaluated the LSTM model, experiments with real-time bandwidth usage prediction were conducted. The objective was to integrate the real-time prediction mechanism with a Software-Defined Networking (SDN) platform in order to be able to detect future bottlenecks and react proactively to avoid them. SDN is a centrally managed network architecture, where a controller maintains a global view of the network and is able to dynamically reconfigure network devices to meet certain needs.

In this case, the SDN controller was running in the Orchestrator virtual machine and was also in charge of receiving collected data from the routers and using the trained LSTM model to predict future bandwidth consumption on their interfaces. Open vSwitch (OVS) [122] was installed on the router virtual machines. Open vSwitch is a virtual network switch, which uses the OpenFlow protocol to connect to the controller and receive reconfiguration commands. Rsync was used in order to provide the controller with the necessary data for the prediction. Rsync is a Linux tool that can copy files from a remote location to a local one in real time. It transferred to the controller the data collected by the router every 3 seconds. The controller then feeds them to the LSTM model and obtain a future value of the bandwidth usage ratio.

In this test, we only monitored one router (namely router2) whose interfaces were bridged to the OVS switch so that the controller could have a view of the traffic passed through them. For the prediction data, as well as the OpenFlow signaling messages, we opted for an out-of-band approach. In other words, the switch-controller communication and the prediction data used a different dedicated channel and not the bridged interfaces used for application data.

A separate instance of the prediction mechanism was running for each one of router2's interfaces. With this setup, the controller received prediction data every 3 seconds

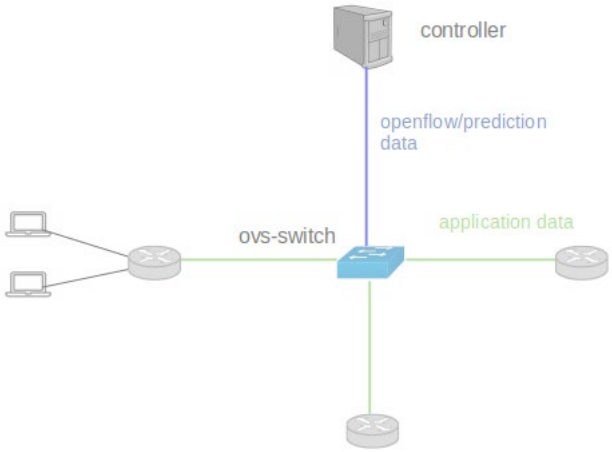

Figure 7.7: Leveraging SDN for Proactive Management of the Network.

and was able to predict each interface's bandwidth 15 seconds ahead. As a prevention measure we experimented with two solutions. A radical one was to block traffic if the bandwidth prediction exceeded a certain threshold. For example, if the prediction for an interface exceeded 0.8 (80% of the link's capacity), the controller instructed the switch to drop new incoming packets from this interface to avoid link overload. A more sophisticated approach was load-balancing the traffic upon reaching a certain threshold. For instance, if the outgoing bandwidth reached 50% of the capacity of a certain link, the controller installed flow rules on the switch that spread traffic across other available links.

7.6 Conclusions and Future Work

Three machine learning algorithms (ARIMA, MLP, and LSTM) were tested to predict bandwith usage ratios 15 seconds ahead. LSTM models have shown their ability to predict network link usage with high accuracy (errors below 3%). These models can be deployed in real time and synchronized with a SDN platform to prevent congestion before it occurs. This type of architecture allows the use of load balancing mechanisms to avoid packet loss by maximizing network throughput.

It is possible that we could get similar results (or even improve them) by removing some features, especially those collected by Dstat that do not pertain to network traffic. To that end, it would be interesting to explore dimensionality reduction methods or simply try smaller feature sets and evaluate their performance. Less features would mean less resources required for prediction, as well as less bandwidth needed to upload the prediction data to the controller.

Investigating the predictability of other parameters might indicate congestion or other network problems. An example would be prediction of future RAM and CPU usage on the router. Resource depletion of network equipment would mean lost packets

and high latency which in turn would cause congestion and degradation of the network performance in general.

Finally, we also plan to migrate to a different testbed network possibly with real hardware in order to re-evaluate the performance and make sure that our solution was not somehow topology dependent. For the same reasons, we also plan to use real applications to generate traffic like FTP, HTTP, and RTP or a real traffic generator.

Chapter 8

Conclusions and future work

8.1 Summary and Conclusions

In this thesis, we proposed efficient techniques to deploy anomaly-based IDSs on real networks. We started by conducting a comprehensive state of the art on neural networks classifiers for KDD Cup 99 and NSL-KDD. This survey determined that the two types of neural networks that perform best on this problem are MLPs and SOMs. It also highlighted several areas for improvement in terms of parameter optimization and data processing.

These findings were used to design two solutions on the same datasets. The first solution uses only neural networks and is based on a three-step process: 1/ data augmentation to rebalance the datasets; 2/ hyperparameters optimization to improve the performance of neural networks and 3/ ensemble learning to maximize the quality of detection. A cascade-structured architecture is adopted to decrease the FPR and increase the TPR by creating a succession of meta-specialists (88.39% accuracy and 1.94% FPR on NSL-KDD). Moreover, the second solution adds other machine learning models that are also optimized and combined to produce meta-specialists. This solution doesn't use a cascade-structured architecture but classifies the connections according to the surest meta-specialist in its prediction (86.59% accuracy and 1.66% FPR on NSL-KDD).

Nonetheless, the development of IDSs with supervised learning did not fulfil the initial objective of the thesis. Indeed, the architectures described previously must be trained on labelled datasets that correspond to the monitored network. We wanted to maintain their good results while being able to adapt them to new networks. A solution using transfer learning was proposed in order to re-train on new networks models previously trained on the CSE-CIC-IDS2018 dataset. This technique requires more data to be correctly tested and detect all categories of attack. We have developed a way to diversify the contribution of anomaly-based IDS, by generating signatures from the detected anomalies. This system can populate signature databases for widely used misuse-based IDSs such as Snort and Suricata. It can also be used as part of a hybrid IDS, combining signature-based and anomaly-based detection.

In order to solve the problem of deploying IDSs on unknown networks, we have

developed an IDS with unsupervised learning. This model uses an innovative protocol analysis architecture to output an anomaly score for the entire packet. Anomaly scores are then averaged to monitor the evolution of abnormal behaviour on the network. An attack is detected when this averaged score exceeds a certain threshold, which allows for true intrusion detection and not packet or flow classification. This system identifies 7 of the 11 unknown attacks on the CICIDS2017 dataset without any false positives.

The last work carried out during this thesis focuses on the prediction of denial of service attacks and, by extension, the prediction of network congestion. We generated our own dataset using a simulated network with periods of high network usage. This dataset was then fed to 3 machine learning models (ARIMA, MLP, LSTM) to evaluate their performance. The LSTM model outperforms ARIMA and MLP with very accurate predictions, rarely exceeding a 3% error (40% for ARIMA and 20% for the MLP). Finally, we showed that the proposed solution can be used in real time with a reaction managed by a SDN platform.

8.2 Future Research Proposals

Different research proposals can be identified for future work in network intrusion detection. Several strategies were proposed during this thesis depending on the type of learning used. Each of these strategies can be improved by proposing solutions to address their weaknesses.

First, the contribution of supervised learning appears to be limited for intrusion detection due to the lack of labelled data. Machine learning techniques like VAEs or GANS could be used to generate labeled traffic containing fake attacks. These models would input normal traffic from the network to be monitored, and output traffic with realistically added attacks. These machine learning algorithms would then be trained on a series of labeled datasets, in order to learn how to insert the same attacks into normal traffic. Ideally, a model would be trained per attack or per type of attack to ensure proper learning and the ability to modify attack ratios in the generated traffic.

The problem with this solution is that the IDS would still not be trained to detect unknown attacks. It would learn attacks from the artificially generated traffic, and thus indirectly from the datasets that were used to train the generative model. However, this technique remains important for validating the performance of a deployed IDS, regardless of the type of learning. This is indeed a problem that can also occur with unsupervised detection: since the IDS has never seen any attacks on the monitored network, we cannot be sure that it detects them correctly. A generative model would allow us to test the performance of the IDS and identify malfunctions before actual deployment.

Another problem that this generative model should solve is the consistency in the sequence of events of the simulated attacks. For example, data extraction traffic cannot occur before the system from which the data is extracted has been compromised. A realistic sequence of events within an attack is particularly important for context-based anomaly detection techniques. This is also an important point to motivate work focused on attack detection, not packet or flow classification. The performance of unsupervised

detection can be improved, especially through a more intelligent aggregation of the different anomaly scores.

Bibliography

[1] M. Roesch, "Snort - lightweight intrusion detection for networks," in Proceedings of the 13th USENIX Conference on System Administration, LISA '99, (USA), p. 229–238, USENIX Association, 1999.

[2] "Suricata | open source ids / ips / nsm engine," Dec. 2009. [Online; accessed 2019-12-16].

[3] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," ACM Trans. Inf. Syst. Secur., vol. 3, p. 186–205, Aug. 2000.

[4] M. Labonne, A. Olivereau, and D. Zeghlache, "A survey of neural network classifiers for intrusion detection (submitted)," 2020.

[5] M. Labonne, A. Olivereau, and D. Zeghlache, "Automatisation du processus d'entraînement d'un ensemble d'algorithmes de machine learning optimisés pour la détection d'intrusion," in Proceedings of Journées C&ESAR 2018, pp. 1–10, Journées C&ESAR, 11 2018.

[6] M. Labonne, A. Olivereau, B. Polvé, and D. Zeghlache, "A cascade-structured meta-specialists approach for neural network-based intrusion detection," in 16th IEEE Annual Consumer Communications & Networking Conference, CCNC 2019, Las Vegas, NV, USA, January 11-14, 2019, pp. 1–6, 2019.

[7] M. Labonne, B. Polvé, and A. Olivereau, "Procédé et système de détection d'anomalie dans un réseau de télécommunications," 2019.

[8] M. Labonne, A. Olivereau, B. Polve, and D. Zeghlache, "Unsupervised protocol-based intrusion detection for real-world networks," in ICNC 2020: International Conference on Computing, Networking and Communications, 2020 International Conference on Computing, Networking and Communications (ICNC), (Big Island, United States), pp. 299–303, IEEE, Feb. 2020.

[9] M. Labonne, C. Chatzinakis, and A. Olivereau 2020.

[10] "Isms family of standards," standard, International Organization for Standardization, Geneva, CH, 2018.

[11] J. P. Anderson, "Computer security technology planning study," Oct. 1972. [Online; accessed 2019-10-18].

[12] J. P. Anderson, "Computer security threat monitoring and surveillance," Feb. 1980. [Online; accessed 2019-10-18].

[13] D. E. Denning, "An intrusion-detection model," IEEE Trans. Softw. Eng., vol. 13, pp. 222–232, Feb. 1987.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python ," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," arXiv:1603.04467 [cs], 3 2016. arXiv: 1603.04467.

[16] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, and Z. Lin, "Automatic differentiation in pytorch," p. 4.

[17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," ACM SIGKDD Explorations Newsletter, vol. 11, p. 10, 11 2009.

[18] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," CoRR, vol. abs/1903.02460, 2019.

[19] J. Song, H. Takakura, Y. Okabe, M. Eto, D. Inoue, and K. Nakao, "Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation," in Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS '11, (New York, NY, USA), p. 29–36, Association for Computing Machinery, 2011.

[20] A. Sperotto, R. Sadre, F. van Vliet, and A. Pras, "A labeled data set for flow-based intrusion detection," in IP Operations and Management (G. Nunzi, C. Scoglio, and X. Li, eds.), (Berlin, Heidelberg), pp. 39–50, Springer Berlin Heidelberg, 2009.

[21] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," 11 2015.

[22] "Kdd-cup-99 task description." [Online; accessed 2017-06-18].

[23] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," IEEE, 2009. [Online; accessed 2017-06-24].

[24] "Canadian institute for cybersecurity | research | datasets | unb." [Online; accessed 2017-07-17].

[25] A. Habibi Lashkari, G. Draper Gil, M. Mamun, and A. Ghorbani, "Characterization of tor traffic using time based features," pp. 253–262, 01 2017.

[26] R. Panigrahi and S. Borah, "A detailed analysis of cicids2017 dataset for designing intrusion detection systems," vol. 7, pp. 479–482, 01 2018.

[27] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," pp. 108–116, 01 2018.

[28] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in Proceedings of COMPSTAT'2010, p. 177–186, Springer, 2010. [Online; accessed 2017-07-21].

[29] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," Journal of Machine Learning Research, vol. 12, no. Jul, p. 2121–2159, 2011.

[30] M. D. Zeiler, "Adadelta: An adaptive learning rate method," arXiv:1212.5701 [cs], 12 2012. arXiv: 1212.5701.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980 [cs], 12 2014. arXiv: 1412.6980.

[32] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," Neural computation, vol. 18, no. 7, p. 1527–1554, 2006.

[33] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10, (USA), pp. 807–814, Omnipress, 2010.

[34] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," Journal of Machine Learning Research, vol. 15, no. 1, p. 1929–1958, 2014.

[35] F. Palenzuela, M. Shaffer, M. Ennis, J. Gorski, D. McGrew, D. Yowler, D. White, L. Holbrook, C. Yakopcic, and T. M. Taha, "Multilayer perceptron algorithms for cyberattack detection," pp. 248–252, 2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS), 7 2016.

[36] N. Mowla, I. Doh, and K. Chae, "Evolving neural network intrusion detection system for mcps," pp. 183–187, 2017 19th International Conference on Advanced Communication Technology (ICACT), 2 2017.

[37] S. Potluri and C. Diedrich, "Accelerated deep neural networks for enhanced intrusion detection system," pp. 1–8, 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), 9 2016.

[38] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," p. 153–160, 2007. [Online; accessed 2017-07-21].

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," arXiv:1502.01852 [cs], 2 2015. arXiv: 1502.01852.

[40] J. Kim, N. Shin, S. Y. Jo, and S. H. Kim, "Method of intrusion detection using deep neural network," pp. 313–316, 2017 IEEE International Conference on Big Data and Smart Computing (BigComp), 2 2017.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," p. 1097–1105, 2012. [Online; accessed 2017-05-29].

[42] A. Ng, "Sparse autoencoder," CS294A Lecture notes, vol. 72, no. 2011, p. 1–19, 2011.

[43] Q. Niyaz, A. Javaid, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," vol. 15, p. 21–26, 2016. [Online; accessed 2017-04-04].

[44] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: transfer learning from unlabeled data," p. 759–766, ACM, 2007. [Online; accessed 2017-04-07].

[45] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," pp. 3854–3861, 2017 International Joint Conference on Neural Networks (IJCNN), 5 2017.

[46] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016.

[47] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?," Journal of Machine Learning Research, vol. 11, no. Feb, p. 625–660, 2010.

[48] K. Alrawashdeh and C. Purdy, "Toward an online anomaly intrusion detection system based on deep learning," pp. 195–200, 2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA), 12 2016.

[49] Y. Liu and X. Zhang, "Intrusion detection based on idbm," pp. 173–177, 2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech), 8 2016.

[50] M. Z. Alaya, S. Bussy, S. Gaïffas, and A. Guilloux, "Binarsity: a penalization for one-hot encoded features," arXiv:1703.08619 [stat], 3 2017. arXiv: 1703.08619.

[51] "Deep neural network - file exchange - matlab central." [Online; accessed 2018-07-31].

[52] "Deep learning toolbox - file exchange - matlab central." [Online; accessed 2018-07-31].

[53] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," IEEE Transactions on Neural Networks, vol. 5, pp. 157–166, 3 1994.

[54] R. C. Staudemeyer, "Applying long short-term memory recurrent neural networks to intrusion detection," South African Computer Journal, vol. 56, no. 1, p. 136–154, 2015.

[55] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," pp. 1–5, 2016 International Conference on Platform Technology and Service (PlatCon), 2 2016.

[56] T. T. H. Le, J. Kim, and H. Kim, "An effective intrusion detection classifier using long short-term memory with gradient descent optimization," pp. 1–6, 2017 International Conference on Platform Technology and Service (PlatCon), 2 2017.

[57] "Ghsom - the growing hierarchical self-organizing map." [Online; accessed 2017-07-21].

[58] A. Rauber, D. Merkl, and M. Dittenbach, "The growing hierarchical self-organizing map: exploratory analysis of high-dimensional data," IEEE Transactions on Neural Networks, vol. 13, pp. 1331–1341, 11 2002.

[59] T. Kiziloren and E. Germen, "Anomaly detection with self-organizing maps and effects of principal component analysis on feature vectors," vol. 2, pp. 509–513, 2009 Fifth International Conference on Natural Computation, 8 2009.

[60] S. McElwee and J. Cannady, "Improving the performance of self-organizing maps for intrusion detection," pp. 1–6, SoutheastCon 2016, 3 2016.

[61] V. Almendra and D. Enachescu, "Using self-organizing maps for binary classification with highly imbalanced datasets," International Journal of Numerical Analysis and Modeling, series b, vol. 5, no. 3, p. 238–254, 2014.

[62] X. Cheng and S. Wen, "A real-time hybrid intrusion detection system based on principle component analysis and self organizing maps," vol. 3, pp. 1182–1185, 2010 Sixth International Conference on Natural Computation, 8 2010.

[63] D. Ippoliti and X. Zhou, "An adaptive growing hierarchical self organizing map for network intrusion detection," pp. 1–7, 2010 Proceedings of 19th International Conference on Computer Communications and Networks, 8 2010.

[64] M. Salem and U. Buehler, "An enhanced ghsom for ids," pp. 1138–1143, 2013 IEEE International Conference on Systems, Man, and Cybernetics, 10 2013.

[65] P. Yichun, N. Yi, and H. Qiwei, "Research on intrusion detection system based on irbf," pp. 544–548, 2012 Eighth International Conference on Computational Intelligence and Security, 11 2012.

[66] M. Dorigo and M. Birattari, "Ant colony optimization," in Encyclopedia of machine learning, p. 36–39, Springer, 2011.

[67] G. A. Carpenter, "Art 2-a: An adaptive resonance algorithm for rapid category learning and recognition," Neural networks, 1991. [Online; accessed 2017-07-26].

[68] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy art: Fast stable learning and categorization of analog patterns by an adaptive resonance system," Neural networks, vol. 4, no. 6, p. 759–771, 1991.

[69] X. Han, "An improved intrusion detection system based on neural network," vol. 1, pp. 887–890, 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems, 11 2009.

[70] N. Ngamwitthayanon and N. Wattanapongsakorn, "Fuzzy-art in network anomaly detection with feature-reduction dataset," pp. 116–121, 7th International Conference on Networked Computing, 9 2011.

[71] P. Mell, V. Hu, R. Lippmann, J. Haines, and M. Zissman, An overview of issues in testing intrusion detection systems. 2003. [Online; accessed 2017-07-25].

[72] R. Werlinger, K. Hawkey, K. Muldner, P. Jaferian, and K. Beznosov, "The challenges of using an intrusion detection system: is it worth the effort?," p. 107–118, ACM, 2008. [Online; accessed 2017-07-25].

[73] P. Chifflier and A. Fontaine, "Architecture système sécurisée de sonde ids réseau," (Rennes, France), pp. 97–109, Journées C&ESAR 2014 : Détection et réaction face aux attaques informatiques, 2014. [Online; accessed 2017-02-24].

[74] M. Z. Alaya, S. Bussy, S. Gaïffas, and A. Guilloux, "Binarsity: a penalization for one-hot encoded features in linear supervised learning," Journal of Machine Learning Research, vol. 20, no. 118, pp. 1–34, 2019.

[75] J. Zhang and I. Mani, "KNN Approach to Unbalanced Data Distributions: A Case Study Involving Information Extraction," in Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Datasets, 2003.

[76] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," IEEE Trans. Syst. Man Cybern., vol. 2, pp. 408–421, 1972.

[77] I. Tomek, "An experiment with the edited nearest-neighbor rule," IEEE Transactions on Systems and Man and Cybernetics, vol. 6, no. 6, pp. 448–452, 1976.

[78] M. R. Smith, T. R. Martinez, and C. G. Giraud-Carrier, "An instance level analysis of data complexity," Mach. Learn., vol. 95, no. 2, pp. 225–256, 2014.

[79] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, 2002.

[80] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in IN: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IEEE WORLD CONGRESS ON COMPUTATIONAL INTELLIGENCE), IJCNN 2008, pp. 1322–1328, 2008.

[81] I. Tomek, "Two Modifications of CNN," IEEE Transactions on Systems, Man, and Cybernetics, vol. 7(2), pp. 679–772, 1976.

[82] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," J. Mach. Learn. Res., vol. 18, pp. 559–563, Jan. 2017.

[83] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," CoRR, vol. abs/1603.04467, 2016.

[84] F. Chollet et al., "Keras." `https://keras.io`, 2015.

[85] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS'11, (USA), pp. 2546–2554, Curran Associates Inc., 2011.

[86] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," J. Mach. Learn. Res., vol. 13, pp. 281–305, Feb. 2012.

[87] J. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms."

[88] T. G. Dietterich, "Ensemble methods in machine learning," in Proceedings of the First International Workshop on Multiple Classifier Systems, MCS '00, (London, UK, UK), pp. 1–15, Springer-Verlag, 2000.

[89] B. Adhi Tama, A. Patil, and K. H. Rhee, "An improved model of anomaly detection using two-level classifier ensemble," 08 2017.

[90] T. Pham, E. Foo, S. Suriadi, H. Jeffrey, and H. Lahza, "Improving performance of intrusion detection system using ensemble methods and feature selection," pp. 1–6, 01 2018.

[91] N. Kanakarajan and K. Muniasamy, Improving the Accuracy of Intrusion Detection Using GAR-Forest with Feature Selection, pp. 539–547. 10 2016.

[92] M. S. Pervez and D. Farid, "Feature selection and intrusion classification in nsl-kdd cup 99 dataset employing svms," SKIMA 2014 - 8th International Conference on Software, Knowledge, Information Management and Applications, 04 2015.

[93] S. Nembrini, I. König, and M. Wright, "The revival of the gini importance?," Bioinformatics (Oxford, England), vol. 34, 05 2018.

[94] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," CoRR, vol. abs/1808.01974, 2018.

[95] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," CoRR, vol. abs/1903.02460, 2019.

[96] S. Zanero and S. M. Savaresi, "Unsupervised learning techniques for an intrusion detection system," in Proceedings of the 2004 ACM Symposium on Applied Computing, SAC '04, (New York, NY, USA), pp. 412–419, ACM, 2004.

[97] S. Amante, J. Rajahalme, B. E. Carpenter, and S. Jiang, "IPv6 Flow Label Specification," Tech. Rep. 6437, Nov. 2011.

[98] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[99] G. Gu, P. Fogla, D. Dagon, W. Lee, and B. Skorić, "Measuring intrusion detection capability: An information-theoretic approach," in Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ASIACCS '06, (New York, NY, USA), pp. 90–101, ACM, 2006.

[100] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, pp. 1–6, July 2009.

[101] K. Leung and C. Leckie, "Unsupervised anomaly detection in network intrusion detection using clusters.," pp. 333–342, 01 2005.

[102] G. Kotani and Y. Sekiya, "Unsupervised scanning behavior detection based on distribution of network traffic features using robust autoencoders," in 2018 IEEE International Conference on Data Mining Workshops (ICDMW), pp. 35–38, Nov 2018.

[103] A. H. Mirza and S. Cosan, "Computer network intrusion detection using sequential lstm neural networks autoencoders," in 2018 26th Signal Processing and Communications Applications Conference (SIU), pp. 1–4, May 2018.

[104] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," pp. 108–116, 01 2018.

[105] N. Z. Bawany, J. A. Shamsi, and K. Salah, "Ddos attack detection and mitigation using sdn: Methods, practices, and solutions," Arabian Journal for Science and Engineering, vol. 42, pp. 425–441, Feb 2017.

[106] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin, "Simple network management protocol (snmp)," RFC 1157, RFC Editor, United States, 05 1990.

[107] R. Sommer and A. Feldmann, "Netflow: Information loss or win?," in Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment, IMW '02, (New York, NY, USA), pp. 173–174, ACM, 2002.

[108] P. Phaal, S. Panchen, and N. McKee, "Inmon corporation's sflow: A method for monitoring traffic in switched and routed networks," RFC 3176, RFC Editor, United States, 09 2001.

[109] C. So-In, "A survey of network traffic monitoring and analysis tools," p. 24, 01 2006.

[110] M. Welzl, Congestion control principles, ch. 2, pp. 7–53. John Wiley and Sons, Ltd, 2006.

[111] M. Devare and A. Kumar, "Clustering and classification (time series analysis) based congestion control algorithm: Data mining approach," in IJCSNS International Journal of Computer Science and Network Security, vol. 7, pp. 241–246, September 2007.

[112] P. Singhal and A. Yadav, "Congestion detection in wireless sensor network using neural network," in International Conference for Convergence for Technology-2014, pp. 1–4, April 2014.

[113] T. Issariyakul and E. Hossain, Introduction to Network Simulator NS2. Springer Publishing Company, Incorporated, 1st ed., 2010.

[114] J. B. Madalgi and S. A. Kumar, "Congestion detection in wireless sensor networks using mlp and classification by regression," in 2017 3rd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), pp. 226–231, Dec 2017.

[115] J. B. Madalgi and S. Anupama Kumar, "Development of wireless sensor network congestion detection classifier using support vector machine," in 2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS), pp. 187–192, Dec 2018.

[116] N. Zhang, F. Jaafar, and Y. Malik, "Low-rate dos attack detection using psd based entropy and machine learning," in 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), pp. 59–62, June 2019.

[117] "iPerf - iPerf3 and iPerf2 user documentation."

[118] C. Schmoll and S. Zander, "Netmate-user and developer manual," 01 2004.

[119] "dstat(1) - Linux man page."

[120] Y. Bengio and Y. Grandvalet, "No unbiased estimator of the variance of k-fold cross-validation," J. Mach. Learn. Res., vol. 5, pp. 1089–1105, Dec. 2004.

[121] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Comput., vol. 9, pp. 1735–1780, Nov. 1997.

[122] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, NSDI'15, (Berkeley, CA, USA), pp. 117–130, USENIX Association, 2015.

TELECOM SudParis

Titre : Détection d'intrusion réseau par anomalies avec apprentissage automatique

Mots clés : Détection d'intrusion, apprentissage automatique, réseaux de neurones, cybersécurité, entraînement non-supervisé

Résumé : Ces dernières années, le piratage est devenu une industrie à part entière, augmentant le nombre et la diversité des cyberattaques. Les menaces qui pèsent sur les réseaux informatiques vont des logiciels malveillants aux attaques par déni de service, en passant par le phishing et l'ingénierie sociale. Un plan de cybersécurité efficace ne peut plus reposer uniquement sur des antivirus et des pare-feux pour contrer ces menaces : il doit inclure plusieurs niveaux de défense. Les systèmes de détection d'intrusion (IDS) réseaux sont un moyen complémentaire de renforcer la sécurité, avec la possibilité de surveiller les paquets de la couche 2 (liaison) à la couche 7 (application) du modèle OSI. Les techniques de détection d'intrusion sont traditionnellement divisées en deux catégories : la détection par signatures et la détection par anomalies. La plupart des IDS utilisés aujourd'hui reposent sur la détection par signatures ; ils ne peuvent cependant détecter que des attaques connues. Les IDS utilisant la détection par anomalies sont capables de détecter des attaques inconnues, mais sont malheureusement moins précis, ce qui génère un grand nombre de fausses alertes. Dans ce contexte, la création d'IDS précis par anomalies est d'un intérêt majeur pour pouvoir identifier des attaques encore inconnues.

Dans cette thèse, les modèles d'apprentissage automatique sont étudiés pour créer des IDS qui peuvent être déployés dans de véritables réseaux informatiques. Tout d'abord, une méthode d'optimisation en trois étapes est proposée pour améliorer la qualité de la détection : 1/ augmentation des données pour rééquilibrer les jeux de données, 2/ optimisation des paramètres pour améliorer les performances du modèle et 3/ apprentissage ensembliste pour combiner les résultats des meilleurs modèles. Les flux détectés comme des attaques peuvent être analysés pour générer des signatures afin d'alimenter les bases de données d'IDS basées par signatures. Toutefois, cette méthode présente l'inconvénient d'exiger des jeux de données étiquetés, qui sont rarement disponibles dans des situations réelles. L'apprentissage par transfert est donc étudié afin d'entraîner des modèles d'apprentissage automatique sur de grands ensembles de données étiquetés, puis de les affiner sur le trafic normal du réseau à surveiller. Cette méthode présente également des défauts puisque les modèles apprennent à partir d'attaques déjà connues, et n'effectuent donc pas réellement de détection d'anomalies. C'est pourquoi une nouvelle solution basée sur l'apprentissage non supervisé est proposée. Elle utilise l'analyse de l'en-tête des protocoles réseau pour modéliser le comportement normal du trafic. Les anomalies détectées sont ensuite regroupées en attaques ou ignorées lorsqu'elles sont isolées. Enfin, la détection la congestion réseau est étudiée. Le taux d'utilisation de la bande passante entre les différents liens est prédit afin de corriger les problèmes avant qu'ils ne se produisent.

Title : Anomaly-Based Network Intrusion Detection Using Machine Learning

Keywords : Intrusion detection, machine learning, neural networks, cyber security, unsupervised learning

Abstract : In recent years, hacking has become an industry unto itself, increasing the number and diversity of cyber attacks. Threats on computer networks range from malware to denial of service attacks, phishing and social engineering. An effective cyber security plan can no longer rely solely on antiviruses and firewalls to counter these threats: it must include several layers of defence. Network-based Intrusion Detection Systems (IDSs) are a complementary means of enhancing security, with the ability to monitor packets from OSI layer 2 (Data link) to layer 7 (Application). Intrusion detection techniques are traditionally divided into two categories: signatured-based (or misuse) detection and anomaly detection. Most IDSs in use today rely on signature-based detection; however, they can only detect known attacks. IDSs using anomaly detection are able of detecting unknown attacks, but are unfortunately less accurate, which generates a large number of false alarms. In this context, the creation of precise anomaly-based IDS is of great value in order to be able to identify attacks that are still unknown.

In this thesis, machine learning models are studied to create IDSs that can be deployed in real computer networks. Firstly, a three-step optimization method is proposed to improve the quality of detection: 1/ data augmentation to rebalance the dataset, 2/ parameters optimization to improve the model performance and 3/ ensemble learning to combine the results of the best models. Flows detected as attacks can be analyzed to generate signatures to feed signature-based IDS databases. However, this method has the disadvantage of requiring labelled datasets, which are rarely available in real-life situations. Transfer learning is therefore studied in order to train machine learning models on large labeled datasets, then finetune them on benign traffic of the network to be monitored. This method also has flaws since the models learn from already known attacks, and therefore do not actually perform anomaly detection. Thus, a new solution based on unsupervised learning is proposed. It uses network protocol header analysis to model normal traffic behavior. Anomalies detected are then aggregated into attacks or ignored when isolated. Finally, the detection of network congestion is studied. The bandwidth utilization between different links is predicted in order to correct issues before they occur.

Institut Polytechnique de Paris
91120 Palaiseau, France