FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Online Machine Learning-enabled Network Intrusion Detection

Tiago Miguel Pereira Ribeiro

MASTER DISSERTATION

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Ricardo Santos Morla

Co-Supervisor: Carlos Novo

July 20, 2021

© Tiago Miguel Pereira Ribeiro, 2021

Abstract

Nowadays, the number of attackers using encryption as a means of concealing their activity is growing. Attackers are increasingly using complex and sophisticated systems to obscure their personal agendas, such as Transport Layer Security (TLS).

The introduction of machine learning into cybersecurity has enabled the detection of traffic with encrypted malware. Therefore, this tool can be integrated into intrusion detection systems (IDS). Most intrusion detection systems use a blacklist to determine the existence of malware in traffic. Machine learning allows new horizons to be opened in intrusion detection.

In particular, the evolution of malware may present itself as an obstacle to common detection techniques based on blacklisting. The blacklist is a set of rules that allows the detection system to understand whether the traffic is malignant or benign.

In this thesis we propose to create an architecture that allows the incorporation of machine learning in an intrusion detection system, and in parallel to use the new technology to improve the system's blacklist. This work has as main goal to prove the functionality of the system, and also to evaluate the performance of machine learning in the presence of traffic evolution in time. We proceeded to several evaluations and studies in order to draw conclusions about the adopted architecture, but also to understand and retain knowledge of the evolution of the machine learning component in the face of the variation of traffic characteristics over time.

The results obtained showed that the adopted architecture allows the learning component to adapt to the variations of the traffic characteristics.

Resumo

Atualmente, o número de atacantes a usar encriptação como meio de ocultação da sua atividade está a crescer. Os atacantes recorrem cada vez mais a sistemas complexos e sofisticados para ofuscar as suas agendas pessoais, como é exemplo Transport Layer Security (TLS).

A introdução da aprendizagem computacional na cibersegurança permitiu a detecção de tráfego com malware encriptado. Sendo assim, esta ferramenta poderá ser integrada nos sistemas de deteção de intrusão (IDS). Os sistemas de deteção de intrusão, na sua maioria, recorrem a uma lista de regras para determinarem a existência de malware no tráfego. A aprendizagem computacional permite abrir novos horizontes na detecção de intrusão.

Em particular, a evolução do malware poderá apresentar-se como entrave para as técnicas de deteção comuns, baseadas em lista negra. A lista negra é um conjunto de regras que permite ao sistema de detecção perceber se o tráfego é maligno ou benigno.

Nesta tese propomo-nos a criar uma arquitetura que permita incorporar a aprendizagem computacional num sistema de detecção de intrusão, e em paralelo utilizar a nova tecnologia para melhorar a lista negra do sistema. Este trabalho tem como principal objetivo comprovar a funcionalidade do sistema, e também avaliar o desempenho da aprendizagem computacional na presença de evolução do tráfego no tempo. Nós procedemos a várias avaliações e estudos de forma a retirar conclusões sobre a arquitetura adotada, mas também compreender e reter conhecimento da evolução da componente de aprendizagem computacional face à variação das características do tráfego ao longo do tempo.

Os resultados obtidos mostraram que a arquitetura adotada permite que a componente de aprendizagem se adapte às variações das características do tráfego.

Acknowledgments

First of all I want to thank Professor Ricardo Morla and Co-Supervisor Carlos Novo, from the Faculty of Engineering University of Porto, for letting me be part of your working group and for believing in me and in my abilities. I also thank you for all the work meetings held. I also thank you for the work theme, which always enticed me and helped me to overcome difficulties that arose.

I also thank RODZ, GÁBz RABZ and POLÉ for keeping up with me during this time of writing, humhum for all the flashes that POLÉ photographed. For all the lovely smokes that RABZ loved and for Mr RODZ's exemplary behaviour at comps.

I thank, Mr. RODZ for all the war days spent at FEUP, as well as confined at home.

To Arminda, I want to express my thanks for all the translation sessions and, above all, for the backrest that put me to sleep for many years.

I thank, my parents for having provided me the opportunity to have studied in a University, for having raised me and put up with me in this journey of my life, thank you.

Finally, I thank Cintia, better known as Daniela by the family, for putting up with me even when I am very annoying XD. For the love that she gives me every day, my many thanks.

Tiago Ribeiro

"Try not to become a man of success, but rather try to become a man of value."

Albert Einstein

Contents

List of Figures

List of Tables

Listings

Acronyms and Abbreviations

API	Application Programming Interface
WWW	*World Wide Web*
IDS	Intrusion Detection System
CIA	Confidentiality, Integrity and Availability
NIDS	Network Intrusion Detection System
HIDS	Host Intrusion Detection System
OISF	Open Information Security Foundation
IPS	Intrusion Prevention System
TLS	Transport Layer Security
SSL	Secure Sockets Layer
LAN	Local Area Network
WAN	Wide Area Network
TCP	Transmission Control Protocol
IP	Internet Protocol
TAP	Terminal Access Point
C2	Command and Control
GAN	Generative Adversarial Networks
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure (HTTP over TLS or SSL)
MTA	Malware Traffic Analysis
CPU	Central Processing Unit
GPU	Graphics Processing Unit
ML	Machine Learning

Chapter 1

Introduction

The first chapter of this document includes a brief introduction to the issue of detecting malware's Command and Control encrypted traffic. The context and motivation behind this dissertation are delineated and summarised in a problem description, and then the objectives, contributions and document structure are presented.

1.1 Context and Motivation

Nowadays, machine learning is increasingly seen to be growing in all existing technologies [8]. The field of cybersecurity is no different, there have been continuous efforts in this direction as this field is becoming more and more complex [9].

In the field of IDS, there is an extensive range of research in the area of machine learning, since the increase of this type of technology enhances various opportunities in this system. One of these is the early detection of new attacks, which the so-called normal systems (the old methods) did not allow [1]. The introduction of machine learning has begun to allow detection of malware command and control traffic over TLS. However, the performance and reliability of the detection system still an unknown subject that has begun to be studied.

Online machine learning introduced in NIDS to control traffic over TLS is a fast-growing area. This project is captivating since it dominates several areas, as well as presenting a challenge to engineering. This project explores the optimisation of high-performance service, evolution and learning techniques [4]. The creation of this project will also fill a gap in the investigation of the performance over time of systems to detect malware command and control traffic over TLS, as well as keeping systems more secure.

Therefore, this area is open to new studies and the development of new methods with projection into the future.

1.2 Problem Description

The main issue is the detection of C2 malware in encrypted traffic. Malware is constantly evolving, and adopting stealthier C2 architectures to avoid detection by blending in with legitimate traffic. On the one hand, it is necessary to determine if malware detector is also evolving and adapting to the evasive tendencies of malware, and design of an architecture that allows the intrusion detection system to evolve and adapt over time.

1.3 Objectives

The goal of this dissertation is to implement an IDS capable of detecting malware command and control traffic over TLS and evolve/adapt to the changes of the traffic over time.

In order to do so, this system will include a) a base component that detects TLS flows with certificates in the blacklist; b) a machine learning component that classifies TLS flows; and c) an information storage component that stores the information about the TLS flows that allows the machine learning model to be re-trained and the blacklist of certificates to be updated.

1.4 Contributions

The main contributions of this dissertation are:

1. Implementation of an IDS capable of detecting malware command and control traffic over TLS and evolve/adapt to the changes of the traffic over the time.

2. Exploring the adaptability of different machine learning methods that help ML models in changing of malware characteristics.

1.5 Document Structure

Apart from this introduction, this document is divided into the following chapters: Chapter 2 - Literature Review is presented the technology to be used, as well as approaches and work similar to the project to be developed; Chapter 3 - System Description, Environment and Implementation is where the complete description of the system will be made, the environment for which it will be developed, as well as the solutions applied in the conception of the system; Chapter 4 - Command and Control Traffic Detection is where the topic of machine learning and the solutions found will be addressed; Chapter 5 - Offline Learning will cover all studies related to the performance and evolution of the model over time; Chapter 6 - Online Learning will cover all studies related to the performance and evolution of the model over time; Chapter 7 - Conclusions discuss the results obtained in the previous chapters and finally highlight the most important conclusions and present proposals for future work.

Chapter 2

Literature Review

The introduction of machine learning in IDS architectures has become increasingly popular and ML technology has grown considerably, especially in the area of malware command and control traffic [10] . However, the maintenance and updating of ML models is a challenge for IDS and IPS [7].

2.1 Intrusion Detection Systems vs. Intrusion Prevention System

Technologies like IDS [11] and IPS [1] allow networks to be monitored.

The performance of these detection mechanisms is extremely important, especially under demanding computing conditions, at around 10 Gbps. It is necessary to choose wisely the decision algorithms to maintain the performance at these levels, as also keeping the false positive and false negative rate low, in order to achieve the best possible product [12].

2.1.1 Intrusion Detection Systems

The IDS has the objective of detecting intrusions in the network, in order to be able to provide security to the system. The monitors analyse the network and detect any malign activity in the network, as can be seen in Figure 2.1. If there is any anomaly, they create an alert for the system admin with detailed information of the problem [13].

2.1.2 Intrusion Prevention System

IPS is much more capable in relation to IDS. IPS not only detects and alerts of an intrusion, but also eliminates that threat, preventing the system from being affected, as can be seen in Figure 2.2. IPS analyses all packets and those that are considered malignant are discarded. However, a small delay is introduced that cannot be avoided because the system has to analyse all the packets and decide which ones are dropped and which ones pass into the secured network [1].

Nevertheless, it is important to remember that if IPS rules are wrongly implemented, some safe traffic may be dropped while an IDS simply generates misleading alerts [1].

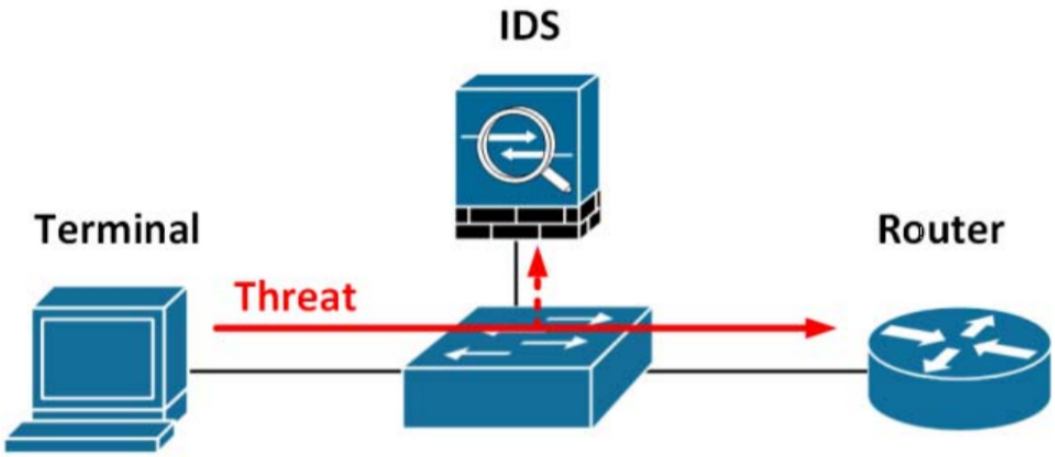


Figure 2.1: Intrusion Detection System logical topology [1]

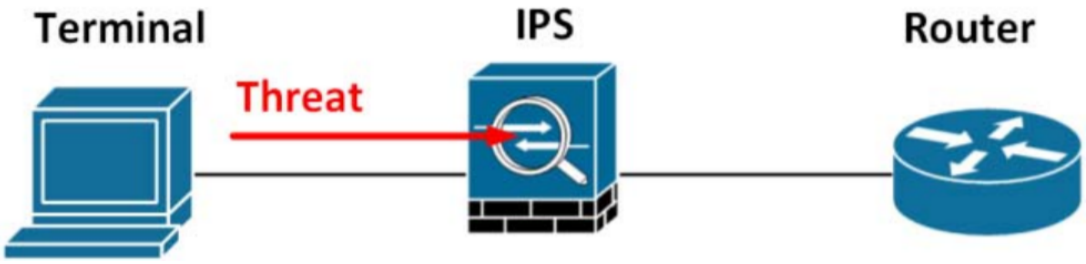


Figure 2.2: Intrusion Prevention System logical topology and usually physical too [1]

2.2 Malware Command and Control Over TLS

Malware is malicious code, which is intended to infiltrate the system in an illicit way to extract information, gain control or even cause damage to the target [14].

This type of software is often used to steal money. As research and analysis show, bank malware has caused damage, such as ZEUS, Citadel, Carberp, SpeEye and Soraya [15]. With this requirement, malware analysis has gained great strength with the aim of minimizing the problems caused by malware, investing more and more in autonomous detection systems (IDS) [16]. Autonomous detection systems must have an extensive knowledge of threats in order to detect and alert the administrator. The problem is the taxonomy of attacks is varied and studies show that IDS identifies only one third of the taxonomy of existing threats. This leads developers to try to choose the best Datasets to train their MLs in order to get the best possible results [14].

Another problem is that malware is already being developed in a way that goes unnoticed, i.e. they are already starting to use methods of anti-analysis [17]. However, defenders begun to use forms of malware detection that are more difficult to violate, thus enabling it to detect malware that is attempting to recreate legitimate traffic [18].

2.2.1 Botnets and Command Control

Botnets are networks formed by computers, called bots. These bots are controlled by botmasters to perform the desired malignant activities. Experts think that there are approximately 16-25% of infected computers belong to a botnet [19].

The description of the C2 operation is summarised in Figures 2.3 and 2.4. The first phase of the attack refers to a malware injection into the host. The second depends on the first injection in order to be able to download and execute the malware that will force the host to behave like a bot. Later on, malware establishes a connection with the command and control centre of the botnet. This is the most delicate part of the operation since this is where connections to C2 can be detected. The fourth phase is the execution of the task to which the bot has been assigned. Finally, the last phase is maintenance and update the platform, it is at this moment that the system is updated in order to add new control tools.

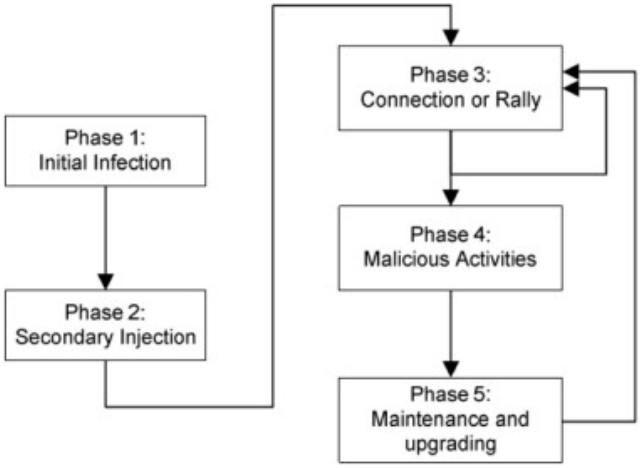


Figure 2.3: The life cycle of a botnet

Botnets represent one of the greatest threats to cybersecurity, therefore led to an increased effort in the detection of Botnets, this detection is done based on the analysis of the behaviour and the flow intervals [20].

There are many ways to hide the flows of botnets. However, currently attackers use encryption tools to hide data from detectors, such as SSL and TLS [10][2]. With this type of attacks, the selection of the flow characteristics is extremely important, since this will be the material that will provide to ML algorithm to classify encrypted traffic (TLS) [21].

2.2.2 C2 Malware Traffic Over TLS

The detection of C2 malware traffic over TLS is important as it prevents the creation of botnets. It also allows breaking the links between the host and the command centre in the most advanced phases of C2 by identifying and eliminating the respective flows [18].

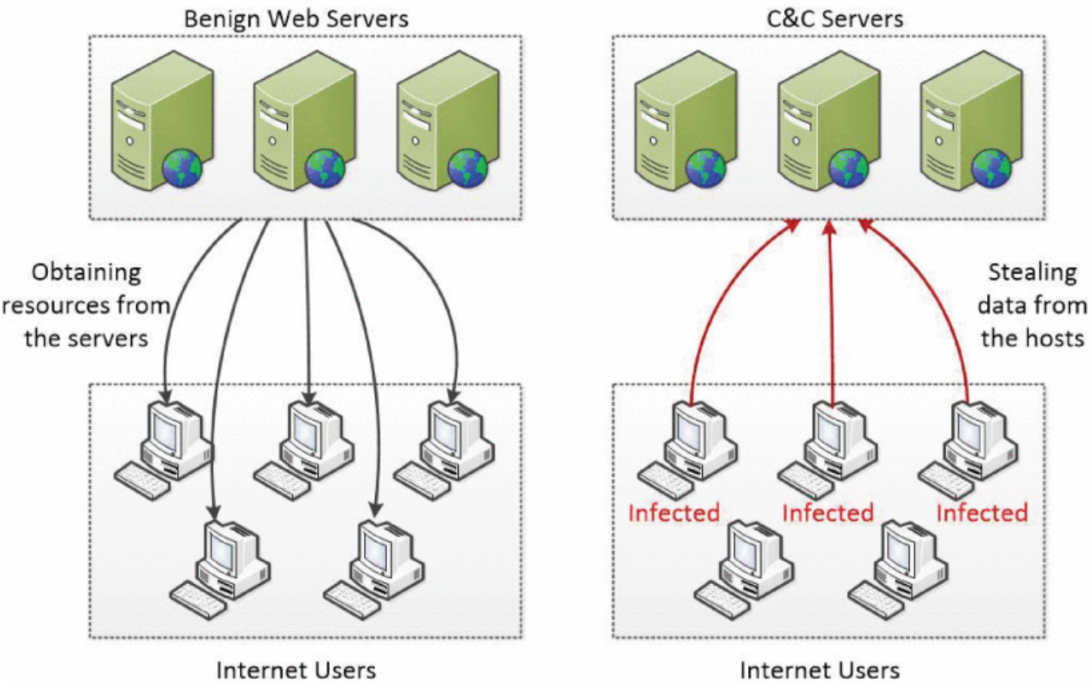


Figure 2.4: Differences between internet users accessing normal servers and C2 servers [2]

The extraction of the correct flow characteristics is extremely important for the classification of encrypted traffic. Some investigations have already concluded that the metadata of client Hello packet in TLS handshake has enough information to distinguish flows with malware or benign traffic [22][23]. In the Table 2.1 are some of the characteristics, that can be evaluated in order to try to find patterns in communications and try to understand the differences between what is benign and what contains malware [2].

Conversely, attackers start using free certificates provided by authorities like Let's Encrypt for their C2 servers or even using self-signed certificates. Since these are easy and quick to create. These certificates usually give errors and warnings in web browsers, but since C2 traffic does not use browsers there is no problem.

Malware authors sometimes use random fields to create the self-signed certificates. This makes their detection easier, as we can see in the C2 traffic captures, as can be seen in Figure 2.5.

2.3 Traffic Detection Using ML

Traffic detection using ML has several stages [24], as can be seen in Figure 2.6. First one must capture traffic. Later, the characteristics of the traffic have to be extracted in order to feed the detection system (ML). These models can classify traffic in defined classes, as can be seen in Figure 2.9.

Table 2.1: Consistency Statistics of Handshake Field [2]

No.	Feature name	Benign channel	Malicious channel
1	Client hello version	94.09%	93.93%
2	Client hello length	26.43%	57.63%
3	Cipher suite number	16.53%	63.70%
4	Client extension number	27.68%	79.41%
5	Client server name	95.32%	99.49%
6	Client session ticket length	41.11%	68.86%
7	Client key exchange length	28.86%	79.24%
8	Client application layer protocol negotiation length	52.53%	89.73%
9	Client signature algorithm number	51.18%	86.44%
10	Client signature algorithm	48.07%	86.39%
11	Client extended master secret	100%	100%
12	Client supported groups	54.98%	89.11%
13	Client padding length	26.86%	86.90%
14	Server hello version	94.08%	94.55%
15	Server hello length	27.52%	78.11%
16	Server cipher suite	72.77%	89.27%
17	Server session ID length	43.60%	84.63%
18	Server extension number	29.60%	78.62%
19	Certificate number	24.09%	79.24%
20	Server compression method	100%	100%

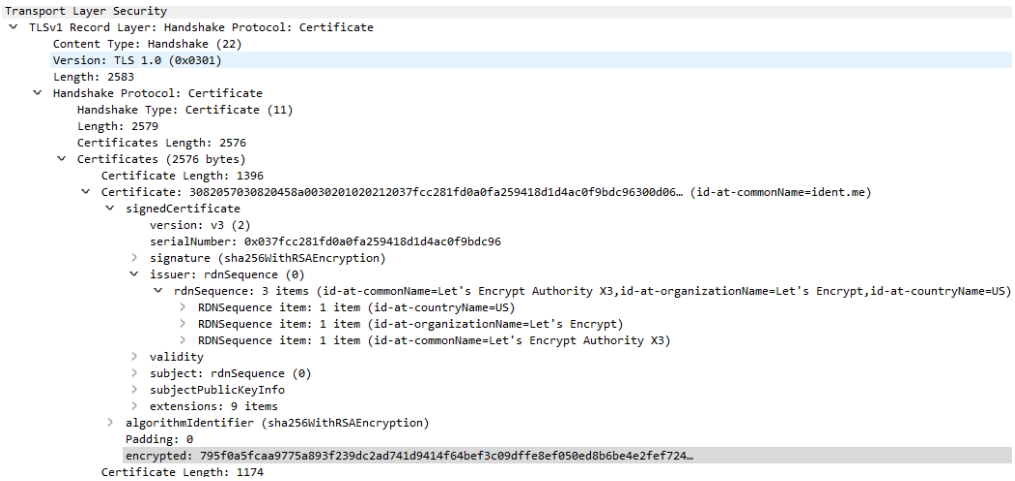


Figure 2.5: Certificate from TLS Malware C2

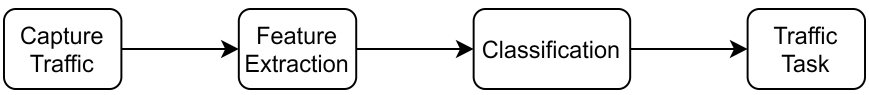


Figure 2.6: Traffic Detector Composition

2.3.1 Features

Using traffic analysis tools it is possible to extract the necessary characteristics to feed the machine learning model. In the case [25], they extracted the data described in Figure 2.7. In other case [3], they extracted the data described in Figure 2.8. In other cases, the authors [2] extract more characteristics, represented in Table 2.1.

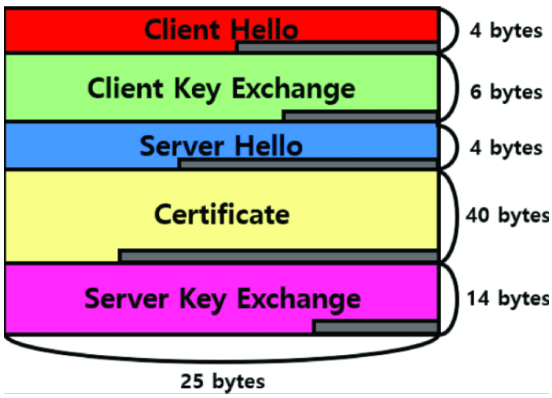


Figure 2.7: TLS flow metadata image format

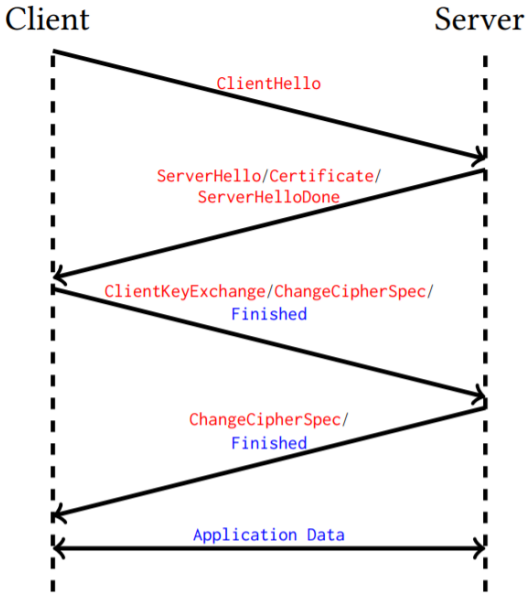


Figure 2.8: Representation of a simplied TLS handshake and application data protocols. The feature sets used in this paper [3] are taken from the unencrypted ClientHello message. Red text represents unencrypted messages, and blue text represents encrypted messages.

In the study of extraction of network traffic characteristics [26], the authors use TSTAT, since it allows to extract information about the flows with extremely efficiency even on a high load traffic.

On other study [23], a random forest classifier feature has been used to determine the model's classification criteria, which makes it possible to determine the importance of a feature in classification. In conclusion, the choice of flow characteristics is very important as it is these that will feed ML [23]. It is important to choose the ones that have the most impact on the classification of the flows, this to improve the classification and at the same time not to fill the classifier with unimportant characteristics. This reduces the quality of the classification as well as reducing the performance of the classifier system. This is because the rating system will be wasting processor time on unimportant features in the rating [27].

2.3.2 Models

The study on machine learning classification models is extensive. There are several classification models, each one having the objective of optimizing certain characteristics. Each model has

its own classification performance for certain characteristics. Like Linear Regression, Logistic Regression, Decision Tree, Random Forest, Support Vector Machine, Multi-layer Perceptron [3] [12].

Another method that started to be used in the feeding of the models is image visualization [4][28][29]. It started to be used to better organize the characteristics, as well as to take advantage of the extensive knowledge about the best models for classification of patterns in images, as it is possible to see in the Figure 2.9. With this, some researchers started to take advantage of this method to classify traffic and malware.

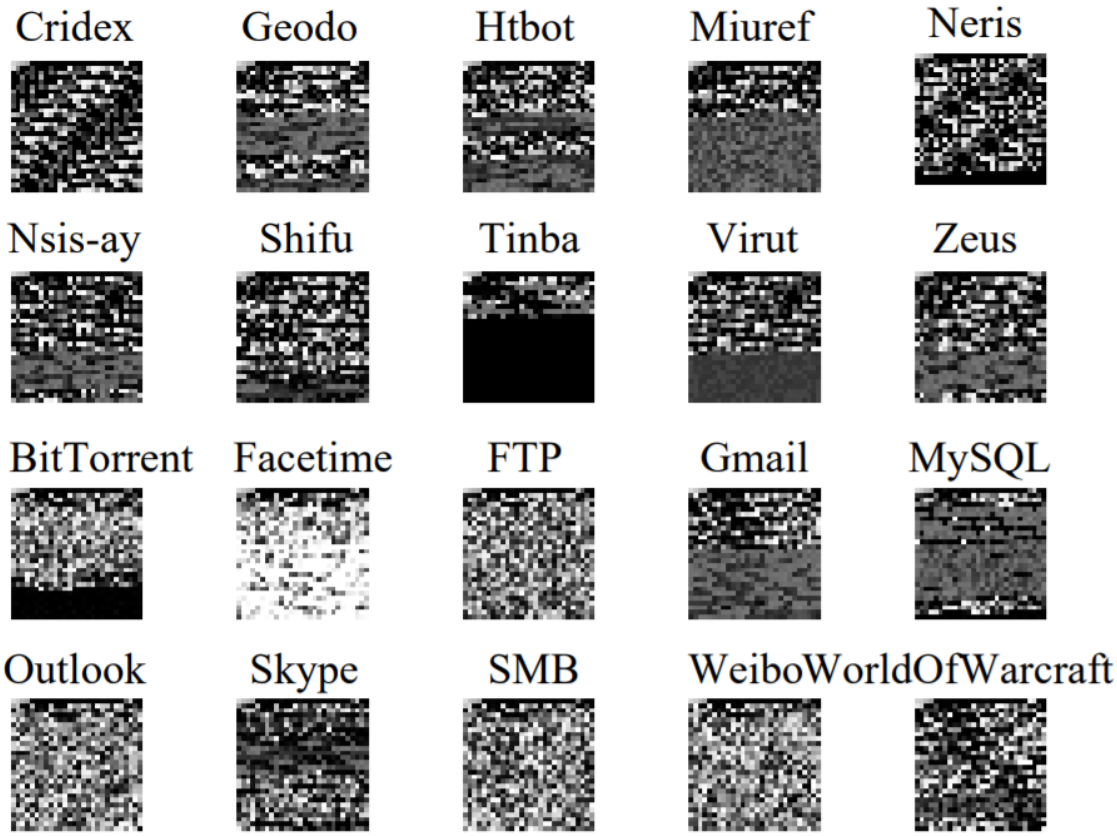


Figure 2.9: Visualization of Classes of Traffic [4]

2.3.3 Detection Tasks

Machine learning detection systems have several applications in different areas. In addition to detecting C2, it can be interesting to detect different objects. Such is the case of detecting the type of application to which that flow belongs [4], Figure 2.9. Another application may be the malware classification [28], after detecting the existence of malware it is possible to realize what kind of malware was detected.

The use of the different machine learning applications can allow a better characterization of the traffic detection as is exemplified above.

2.4 Detection Performance

The metrics of performance evaluation of detectors are important to define the effectiveness of the detection system. This effectiveness is related to high detection rate and low false positive rate, but it should be remembered that there are other types of evaluation factors such as system security, memory, energy consumption, throughput, and much more [14].

2.4.1 Learning Performance

The detector at model level is usually evaluated in terms of accuracy that can be defined in terms: True Positive (TP), True Negative (TN), False Positive (FP) or False Negative (FN). With these terms it is possible to evaluate the model with different metrics: Overall Accuracy (Eq. 2.1), Detection Rates (Eq. 2.2 2.3 2.4 2.5), Precision 2.6, recall, F1 Eq. 2.7, Mcc Eq. 2.8 and much more [14].

$$OverallAccuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.1}$$

$$Sensitivity(akaRecall, TruePositiveRate) = \frac{TP}{TP+FN} \tag{2.2}$$

$$Specificity(akaSelectivity, TrueNegativeRate) = \frac{TN}{TN+FP} \tag{2.3}$$

$$Fallout(akaFalsePositiveRate) = \frac{FP}{TN+FP} \tag{2.4}$$

$$MissRate(akaFalseNegativeRate) = \frac{FN}{TP+FN} \tag{2.5}$$

$$Precision = \frac{TP}{TP+FP} \tag{2.6}$$

$$F1 = \frac{2TP}{2TP+FP+FN} \tag{2.7}$$

$$Mcc = \frac{(TP*TN)-(FP*FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{2.8}$$

2.4.2 Network and Computational Performance

The metrics shown above are intended for the classification component of the detector. The evaluation can also be done globally, evaluating the complete system from start to finish, such as: Packet Loss Ratio (%), Normalised Routing Load (Packets), Average End-End Delay (Seconds), Average Energy Dissipation (Joules), Malicious Drop (Packets), False Detection (%) and Send Buffer Drop (Packets) [30].

A research done on traffic analysis performances, Figure [5], addressed some features extraction software in the market such as: Tstat, Bro, Snort and Suricata. This study compared the four to the level of excitation time, maximum used RAM, I/O rate and average CPU utilisation. It was concluded that Tstat showed the best results, all with the default settings [5].

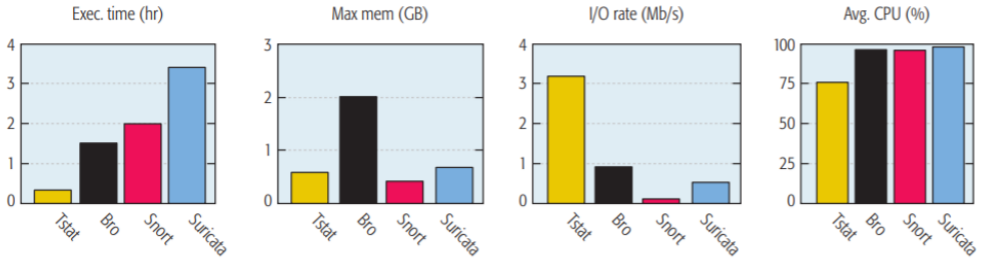


Figure 2.10: STA and NIDS performance comparison [5]

2.5 Training with Live Data

There is a great demand for online ML retraining [31]. There are already incremental learning methods, that are already used to detect this type of traffic [7].

2.5.1 Comparison of Offline and Online Training

In the offline setup a model is created based on a training set. Later, this model classifies a test set of data. Thus, average accuracy is calculated for data not yet seen by the model, Figure 2.11 [6].

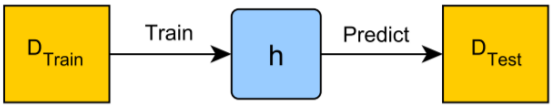


Figure 2.11: Classical scheme of evaluating a batch algorithm in off-line mode [6]

The incremental offline evaluation is somewhat different from the fully offline, as can be seen in Figure 2.12. The evaluation algorithm processes sequentially the training data instead of accessing the training data at once. The process generates multiple models and retrains the model from the previous model, but only the last model is used to make the evaluation through the test data classification. This process is better than the fully offline in that it allows re-training the model from an existing one without having to re-train the whole model. It is very important in Big Data scenarios, allowing us to continuously build a model that is as accurate as possible [6][7].

Data steams use an online configuration, as can be seen in Figure 2.13. The prediction is done according to the previously learned model. Later, the correct classification is revealed and then the loss of the model is calculated, where the model is updated according to the loss value [6] [7].

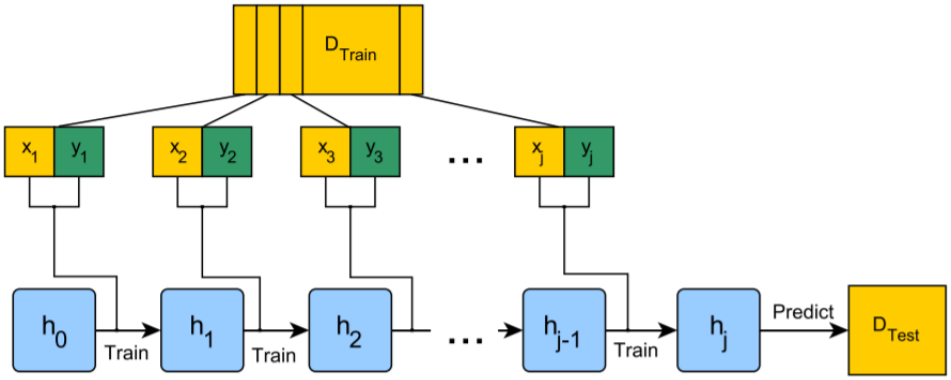


Figure 2.12: The process of testing an incremental algorithm in the off-line setting. Noticeably, only the last constructed model is used for prediction. All data used during training (xi, yi) is obtained from the training set $_{train}$. [6]

The big difference from the previous configuration is that all the models are evaluated in terms of performance, but each one only predicts the next sample. Furthermore, in this configuration there is no disjunction between the training and test data, the sample is used initially to test the model and then to adapt it, while in the others there are different sets of datasets and the test of the model is done only at the end [6][7].

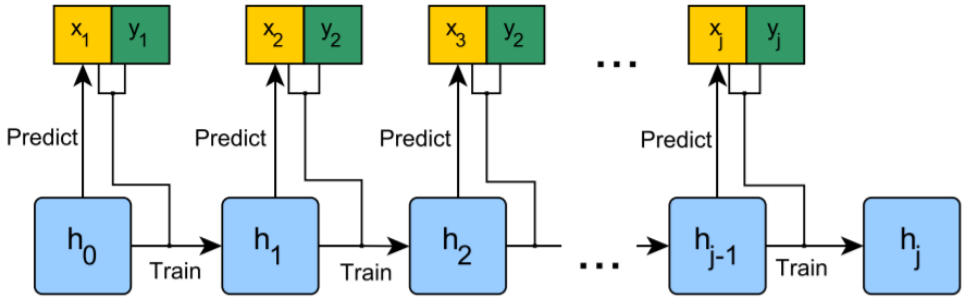
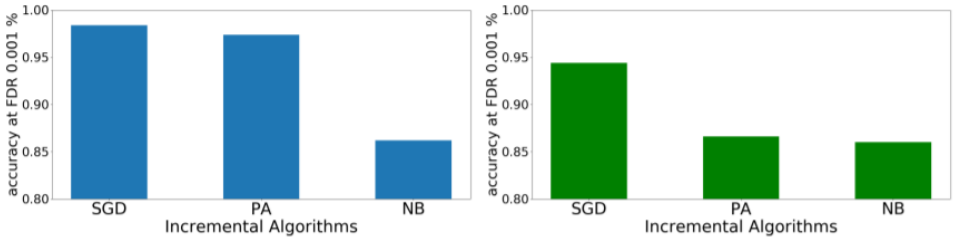


Figure 2.13: The online-learning scheme. Data is not split into training- and testing set. Instead, each model predicts subsequently one example, which is afterwards used for the construction of the next model. [6]

Online accuracy is good for tasks that require an immediate prediction even with few examples provided as well as learn new behaviours that emerge over time (new malware). We conclude that online accuracy converges on average faster than offline accuracy [6][7][31].

The experience [7], concludes that both offline and online training, can detect encrypted anomalies efficiently, Figure 2.14.



(a) off-line accuracy at FDR 0.001% (b) on-line accuracy at FDR 0.001%

Figure 2.14: Results from experiments. [7]

2.6 Conclusion

Malware and C2 have grown and are evolving over time. These types of attacks have looked for new ways to pass unnoticed by detectors. Many attackers have begun to encrypt their activities with TLS to make their detection difficult for machine learning detectors.

As a consequence, IDS have started to use different methods from the usual. They started to adopt machine learning based detectors, since they are much more effective at detecting encrypted traffic compared to conventional detection methods.

Machine learning began to be used in IDS, led to attackers starting to study ways around ML. Attackers start adopting stealthier C2 architectures to avoid detection by blending in with legitimate traffic in order to reduce the performance of classifiers. Thus, detection of C2 is becoming more and more elaborate and difficult. So IDS has to have machine learning methods that overcome these problems. There is also a great computational pressure on the extraction of the flow characteristics and their analysis.

Several studies have been carried out on the training of machine learning models, leading to the appearance of new training methods. However, online learning still has its obstacles [7], it is necessary to make an extended evaluation of the learning influence on the model.

In short, there is still a need for research in the area of command and control traffic over TLS detection and generation. As well as finding efficient methods to set up the whole complex online learning system and evaluate its performance.

Chapter 3

System Description, Environment and Implementation

The normal operation of an IDS relies on rules that are applied to packets and flows in order to detect intrusion and C2 traffic. Our approach is to add an ML classification system in parallel to the normal IDS operation. This will allow us to obtain both outputs — those of the IDS rules and of the ML model.

In this chapter, we will cover the multiple sections of the system, present the design decision taken throughout the development of the system and then we will demonstrate and discuss the results.

3.1 IDS Vs IPS

The distinction from IDS concept to IPS is important since the different approaches have different impacts on the system. IDS analyses the traffic and creates alerts in the presence of malicious traffic according to the system's black list, this system does not prevent the communication of malicious traffic. However, the IPS is similar to the IDS, with the difference that it blocks communications that are considered malicious.

In this thesis, the intrusion detection system was adopted. The IPS blocks malicious communications preventing the collection of these samples, which will be necessary for training the machine learning model, which is one of the objectives of the thesis. Selecting the IDS will imply that the analysed network will be vulnerable to malware. However, the application of this IDS will be for academic purposes and to study the evolution of the computational learning system. This system will serve for analysts and cybersecurity companies to exploit the machine learning systems and update their blacklists.

In short, the system will serve to improve intrusion detection systems. The goal is to apply the acquired knowledge, updated blacklists and detection models to the market of intrusion prevention systems.

3.2 Environment

3.2.1 Place of Operation

Like many IDSs, our system is designed to work between the egress router and the switch of a local network. The IDS system will segregate the internal network, LAN, and the external network, WAN, to capture all traffic coming in and out of the local network. Figure 3.1 shows the location of the IDS in the network. The IDS is designed to only scan TCP packets with TLS (this IDS is targeted for C2 detection), all others that do not fit the profile will not be scanned. It should be remembered that packet forwarding will not be affected, there will be no action on them.

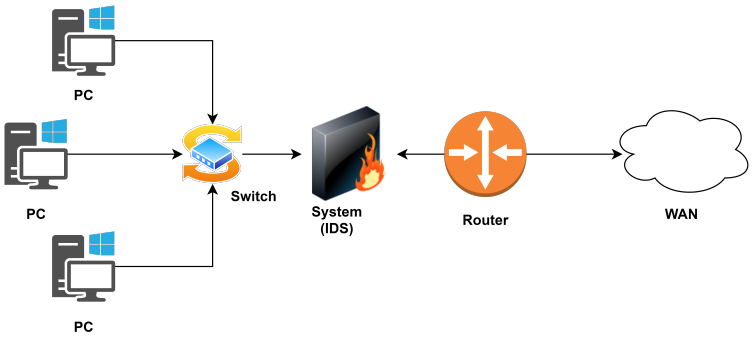


Figure 3.1: Place of operation

3.2.2 Services and Dependencies

Our system was deployed on top of Docker, a containerisation tool. This tool allows us to platform a virtualisation of the entire IDS system. The system virtualisation is an asset to the project since it introduces a greater compartmentalisation of the services in a system. Docker tool increases the security of the system since different services work in different environments, also enables resetting/updating/maintaining services in parallel. Therefore, we created a Docker container with all software, libraries and system configuration files needed. The system was thought and conceived to be portable to any machine and to require as few external dependencies as possible in order to optimise the installation. In short, this approach brings a new set of advantages to the system.

The IDS system is in a **Docker** image with the following dependencies:

- **Docker** — version: 20.10.5 - Docker is a containerisation tool.
- **Jupyter Notebook Server** — version: 6.2.0 - Jupyter Notebook is a tool used on the development of the system.
- **Tensorflow** — version: 2.4.1 - Tensorflow is a machine learning library, that was used for the formation of the classifier.

- **Keras** — version: 2.4.0 - Keras is component of the TensorFlow that allow the train of the machine learning model.

- **Numpy** — version: 1.19.5 - Numpy is library that helps on data treatment.

- **Pandas** — version: 1.1.5 - Pandas is library that helps on data treatment.

- **Tstat** — version: 3.1.1 - Tstat is a tool that was used to extract the characteristics of the TLS flows [32].

- **Suricata** — version: 6.0.2 - Suricata is a tool that was used to extract the fingerprint of the TLS flows [33].

In addition, it was necessary to install the drivers of the network capture card, **Endace DAG**.

3.2.3 Packets-capture

The packet capture is done by a Endace network card. We opted for this network card because it is designed to work with high traffic, but it would be possible to use a normal network card, however the system would be limited by the capacity of the network card.

It is necessary to make the bridge between the capture card and the external programs that will analyse the traffic (Tstat [32] and Suricata [33]). There are three ways to bridge the gap between the capture and analysis software:

1. The first option is a a direct connection between the capture card and the analysis programs. Both Tstat and Suricata allow connection by data streaming, with plugins specially made for Endace boards. Normally when using these boards only one stream is used for the analysis program. Using two programs in the same link could cause reading errors, since there is no documentation regarding this issue, so we consider option one as unfeasible.

2. The second option introduce a larger delay compared to the other two options, since it is necessary to transform the captures from .ref to .pcap and then replicate the traffic to a virtual interface. The second option was also not adopted due to the additional delay.

3. The third option use the network interface. The analysis programs (Tstat and Suricata) read directly from the network interface, with the restriction that this interface must be exposed to the docker container. However, it will be necessary to do Port mirroring on the switch to send a copy of the network packets to the network interface. Solution three was chosen as it is the simplest solution and allows the use of a network card with a single network interface.

3.3 System Description

3.3.1 Detection Assumptions

In this system, the detection of C2 malware encrypted will be done by analysing the characteristics of TCP flows over TLS, so there will be no room for package-by-package analysis, neither there will be any decryption of information.

The assumptions made for the creation of the detector will be summarised in key points to support the concept explored.

- The model will only analyse TCP flows with TLS sections. Since it allows maintaining data privacy protection and at the same time combating the attempt to hide malware in encrypted traffic.
- The detector must be located outside the machine to be analysed.
- The detector aims to understand the communication patterns of malware encrypted sessions trying to pass as benign traffic.
- The detector can only classify the flows that are complete, since it depends on the characteristics of the complete flows for classification.

3.3.2 Intrusion Detection System

The intrusion detection system is made up of three components:

1. **Baseline** is the component that analyses the flows and detects those that have the TLS certificate on the blacklist;
2. **ML component** is an machine learning model that classifies TLS flows, decides if the flow is malignant or not [34];
3. **Acquisition of new information** is a component that stores information about the TLS flows. Later, information saved will allow the ML model to be retrained, as well as updating the black list of TLS certificates.

In the diagram of the Figure 3.2, we can see how the different blocks of the system will be interconnect with each other.

3.3.3 Baseline

The **Baseline** component consists in two python programs. The first Program analyse the TLS flow characteristics (extracted by Tstat) and classify the TLS flows. The second Program analyse logs from first program and fingerprints of TLS flows (extracted by Suricata). By having this information the second program creates alerts (when TLS flows are considered malign) and creates logs for the **Acquisition of new information** component could create training datasets.

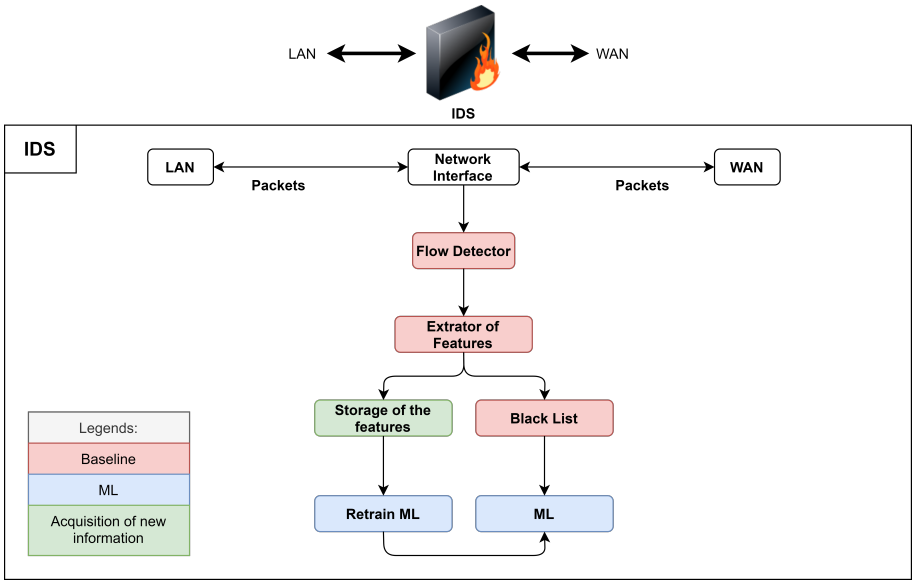


Figure 3.2: Proposed Solution Architecture

The **Baseline** component depends on different types of logs produced by two different analysis tools, Tstat and Suricata. In order to the system work correctly it was necessary to configure the two analysis tools (Tstat and Suricata). Tstat and Suricata were configured to analyse only the TLS flows. Both configurations were made to avoid privacy problems with the inspection of non-encrypted information.

The logs created by Tstat and Suricata are created with a rotation of one hour. Tstat logs will work as a feature extractor that will be fed to the classification system, while the Suricata logs will be used to extract the fingerprints of the flows.

The first program is the one that makes the first actions of the system, which deals with the extraction of the characteristics, the load of the classification model, the classification the of TLS flows and the saving of classifications logs (first program's log). The system has an operator that automatically replaces the loaded model by the most recent one, as soon as it is trained.

Initially, the first program loads the newest classification model, then it will analyse the most recent Tstat's log (complete flows). The Tstat's log contains the characteristics of each TLS flow, each line has about 130 characteristics, from these 130 are removed 88 to feed the classification system. However, before these are sent to the classifier it is necessary to normalise the data, in order to be in conformity with the maximums of the data used on the training of the classification model, this action is described in Section 3.3.5. Therefore, the data is prepared for classification and then classification will take place. The classification value of each flow is added to the log of the first program concatenated with the 88 characteristics used for the classification, as can be seen on Figure 3.6. The operation cycle of the first program is describe on Figure 3.3.

Although, it is good to note that the acquisition of each Tstat line has been implemented with blocking read. The first program use operating system tools to verify the changes of the

file size (Tstat's log), this application allows optimising the processing time in comparison with non-blocking read. It should be mentioned as well that the log created by the first program have a rotation of one hour.

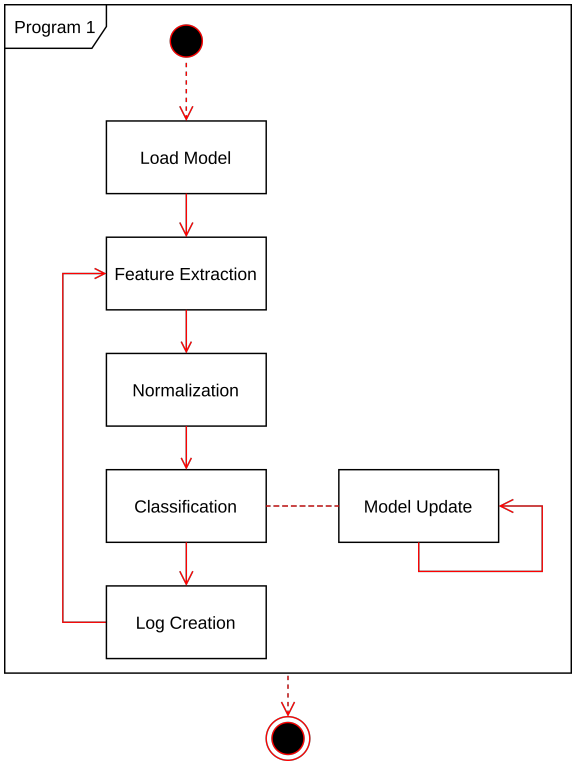


Figure 3.3: Program One

The second program is responsible for creating alerts of TLS flows with malware, by comparing the fingerprints of the TLS flows (Suricata logs) with the blacklist, or by the classification of the information stored in the first program's log. The second program is also responsible for creating a log that are used on creation of datasets for the training of machine learning models. The operation cycle of the second program is simple describe on Figure 3.4

Initially, the program loads the blacklist into a dictionary, use the fingerprint hash to organise the information in order to maximise the performance of the information query. Then it analyses the Suricata's log and extracts the fingerprint of each TLS flow. Later, the program will search in the blacklist the fingerprint of each TLS flow, if it is found an alert will be immediately launched. The second program does not wait for the first program classification to classify the TLS flow since the first program can only classify the flow when a flow ends (a flow can last several minutes). On the contrary, the second program read the Suricata's log that has the analyse of the TLS negotiation and extracts the fingerprint from the certificate, which allows it to detect the threat in advance, minimising the detection time. However, when the threat is not on the list, the second program has to wait for the classification of the first program to be able to analyse the TLS flow.

The next step is to join the information from the first program's log with the Suricata's log. To accomplish this union, two dictionaries must be created, one for each log. The union is made using keys that allow the union (destination IP, destination port, source IP, source port). In Section 4.2 shows one example of a join. This way, it is possible to join the two dictionaries for the creation of the second program's log, which will be extremely important in the creation of training datasets, which in turn will be used in the formulation of future models. In review, the second program's log is characterised by the information from first program's log concatenated with the information taken from the Suricata's log, this structure will be discussed in more detail in the Section 3.3.5. Figure 3.7 and 3.8 show the structure of second program's log.

The second program's log has a rotation of one hour. The second program was implemented with blocking read. However, during the development of the second system it was concluded that Tstat does not identify all TLS flows, anyway Suricata comes to solve this problem since it does not have this problem. In short, we will be able to classify the flows that Tstat don't identify by comparing the fingerprint with the blacklist. However, the TLS flows not identified by Tstat cannot be used in the training of the machine learning as well as the machine learning cannot be used to classify those TLS flows.

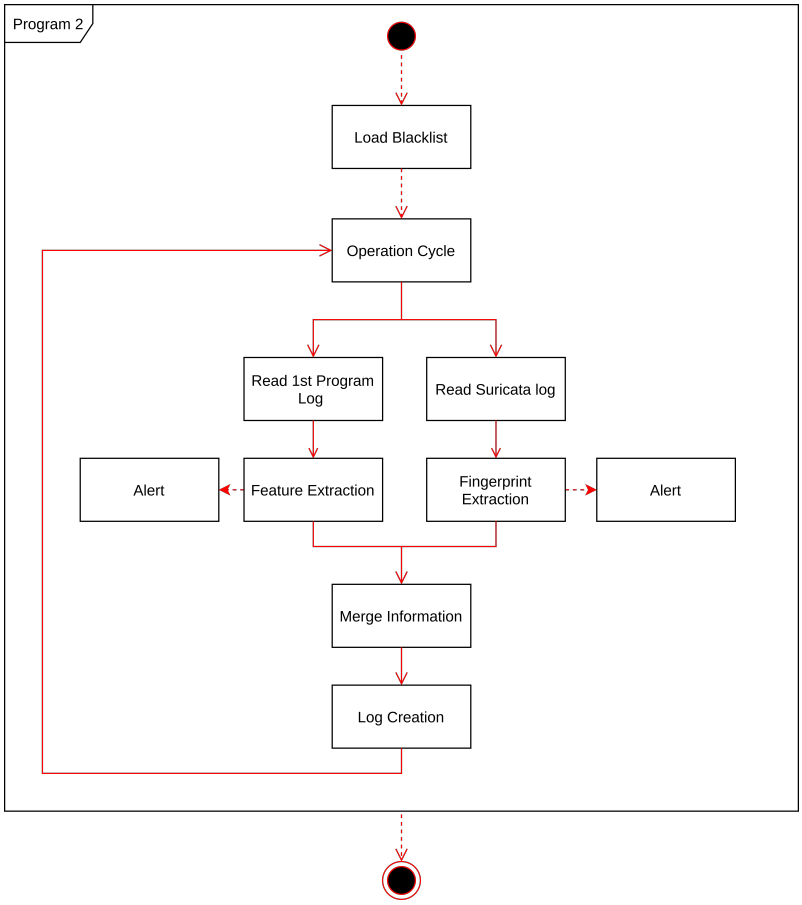


Figure 3.4: Program Two

3.3.4 ML Component

The description of the ML system used will be exclusively demonstrated in Chapter 4. However, it can be advanced with a basic idea of the concept being that the classification has as input a set of 88 characteristics of the TLS flow taken by Tstat.

The intrusion detection system proposed is designed to work with offline training model, but with minor changes it will be possible to adapt the system to an online training system by combining the first program with the second.

The offline training system can run every 15 days, months or years, it uses the logs from the second program. Put all the data together and create a training dataset, this topic will also be covered extensively in the Section 3.3.5 and 4.5.

3.3.5 Acquisition of New Information

The acquisition of new information component is very important since it will bridge the information between the two programs, as it is also allows training the new classification models and updating the black list. The component is divided in three points: the first and second program's logs, the creation of datasets for training and updating the black list.

The program logs are divided into each program. The first program's log is made up as follows, Figure 3.5 and 3.6.

```
1 172.16.141.129 49162 11 0 10 5 1847 4 1847 0 0 0 1 1 185.24.92.229 4743 10 0 10 5 1669 3 1669 0 0 0 1 1
1617618461856.787109 1617618461860.525879 3.739000 0.260000 1.136000 2.858000 3.499000 0.096000 0.308000 1 1 0 0 8192 0
0 0.223663 0.048000 1.055000 0.407702 6 128 128 0.047999 0.048000 0.048000 0.000000 5 128 128 0 0 0 0 0 0 1 0 0 1 0 1460
1205 151 64240 8192 0 1370 151 151 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1460 981 59 64240 64239 0 981 59 981 0 0 0 0 0 0 0 0 0 0
0 --- 4 3 - oolaurauss.sy 0 0 0 1.488000 1.626000 1.850000 3.499000 478 1041 - - 0.0 0.0
```

Figure 3.5: First Program's Log

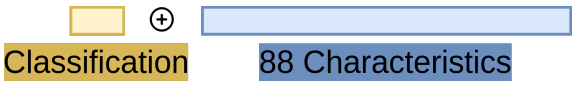


Figure 3.6: First Program's Log Scheme

The second program's log is made up as follows, Figure 3.7 and 3.8.

```
0 172.16.141.129 49162 11 0 10 5 1847 4 1847 0 0 0 1 1 185.24.92.229 4743 10 0 10 5 1669 3 1669 0 0 0 1 1
1615392002362.324951 1615392002366.062988 3.738000 0.260000 1.136000 2.862000 3.499000 0.095000 0.307000 1 1 0 0 8192 0
0 0.214663 0.047000 1.008000 0.389024 6 128 128 0.047599 0.047000 0.048000 0.000548 5 128 128 0 0 0 0 0 0 1 0 0 1 0 1460
1205 151 64240 8192 0 1370 151 151 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1460 981 59 64240 64239 0 981 59 981 0 0 0 0 0 0 0 0 0 0
0 --- 4 3 - oolaurauss.sy 0 0 0 1.487000 1.625000 1.899000 3.499000 478 1041 - - 0.0 0. ef:db:eb:f3:f0:9c:23:87:3a:a2:5c:-
78:cf:5a:ac:95:e2:9e:a7:e6 Dridex C&C
```

Figure 3.7: Second Program's Log

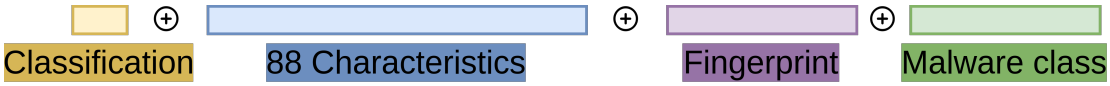


Figure 3.8: Second Program's Log Scheme

The creation of training dataset is done in the following cycle:

1. Searches the storage for the logs of the second program.
2. Loads the chosen data (second program's logs) into a dataframe.
3. Eliminates the unnecessary data, leave only the 88 characteristics of the TLS flow and the classification made through the black list.
4. Loads the matrix with the maximum of each feature and normalises the information.

This dataset can be used to train the ML model or simply saved in a file to be used later. It should be noted that in point 2 of the cycle the logs used for the creation of the dataset must be chosen by the analyst. The system could be completely autonomous if the analyst creat a appropriate rule to choose the logs.

Updating the black list should be done by the analyst as well, simply by downloading an updated black list from a credited source on a weekly basis.

Beside, the analyst could do a more detailed analysis of the second program's log. The analyst have to manually check the flows that the model classified as malware and figure out if the TLS flow were really malware. This action will gadder some fingerprints to the system's personal blacklist that weren't known.

Recall, this process of updating the blacklist is extremely important since it is this action that allows raising the level of reliability of model training, since it will be improving the ground truth.

3.4 Evaluation of the Impact of ML Training

The impact of model training on the system is huge, since it requires a large computational capacity. The solution to alleviate this problem is resort a cluster of GPUs to remove this computational burden from the system. It should be remembered that if there is no GPU in the system or a cluster that the training can resort, the overall system will be severely affected and suffer a CPU starvation, since the training will have to resort to the CPU. Therefore, it is known that both Tstat and Suricata are excessive consumers of the Processing unit [5], Figure 2.10, which indicates the need to use different resources than CPU.

Furthermore, a performance measurement component has been developed to measure the performance of the classifier. The performance measurement component was embedded in the baseline. The component allows to calculate the true/false positives and true/false negatives. The

```
15,0,1,0,2021-04-05 11:52:59.940439
```

Figure 3.9: Program Log Scheme

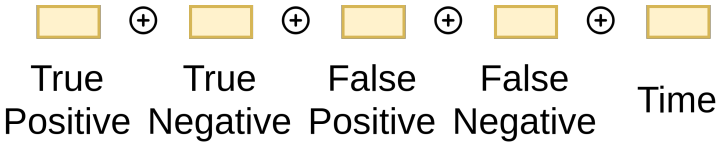


Figure 3.10: Program Two Log Scheme

values are stored in performance's log and are updated every X time defined by the analyst, Figure 3.9 and 3.10.

However, the classification performance (accuracy, recall and f1) will be done in detail , in the Chapter 5,

3.5 Testing and Evaluation of the System

The tests section is the most important since it allows to show the correct functioning of the system, for this a sequence of tests was made to confirm the correct functioning of the system. To empirically verify the correct operation of the system, it was necessary to perform the following experiment.

Key points of the experiment in order to have the conditions properly controlled:

- The captures used well documented (the flows classified by analysts).
- Perform the experiment in a closed environment, virtual network.
- Analyse the logs and verify agreement with the analysis made by the analyst of the captures.

Therefore, test's logs are provided in Appendix A.3.

3.6 Conclusions

In this chapter, the design of the intrusion detection system was carried out. Its architecture was designed, as well as the elements that constitute were constructed and integrated. Therefore, after extensive testing, it was concluded that the chosen architecture works as intended. All the systems that integrate it, except the ML model that will be analysed in the next chapters, passed with distinction and fulfilled their objectives. Thus, we can consider that the intrusion detection system meets the objectives of this thesis.

Chapter 4

Command and Control Traffic Detection

The C2 Malware Traffic Detector is designed to detect encrypted malware over TLS. Malware detectors normally use Deep Packet Inspection techniques. However, in case of encrypted data it is not possible to apply this technique. Therefore, the detector will rely on TLS features to detect the encrypted malware. The design of the Malware C2 Traffic Detector is strongly linked to the continuation of the thesis [34].

This chapter gives background on the C2 classification problem and to the studies carried out in Chapter 5 and 6.

4.1 Malware C2

Malware C2 is dependent on the communication between C2 servers and victims. Therefore, the detector will rely on this dependency (communication) to detect the malware. However, Malware C2 communications have evolved to become undetectable by malware detectors using encrypted communications (TLS).

The detector will use the characteristics of the TLS stream to detect the malware's communication patterns. However, this approach has one problem: it can only detect malware at the end of the communication, since the detector relies on characteristics of the entire communication.

4.2 C2 Traffic Features

Tables 4.1 and 4.2, enumerates the features extracted from the Tstat program. The full description of Tstat's features will be made on the Appendix A.1.

Nevertheless, it is also necessary to know the flow fingerprint to be able to verify if it is malign or benign, for that we resort to a black list. The extraction of this information is done using Suricata, which uses JSON to output its logs.

Table 4.1: Flow Characteristics: Tstat Part One

Core/Basic TCP Set			
C2S	S2C	Short description	Unit
1	15	Client/Server IP addr	-
2	16	Client/Server TCP port	-
3	17	packets	-
4	18	RST sent	0/1
5	19	ACK sent	-
6	20	PURE ACK sent	-
7	21	unique bytes	bytes
8	22	data pkts	-
9	23	data bytes	bytes
10	24	rexmit pkts	-
11	25	rexmit bytes	bytes
12	26	out seq pkts	-
13	27	SYN count	-
14	28	FIN count	-
29		First time abs	ms
30		Last time abs	ms
31		Completion time	ms
32		C first payload	ms
33		S first payload	ms
34		C last payload	ms
35		S last payload	ms
36		C first ack	ms
37		S first ack	ms
38		C Internal	0/1
39		S Internal	0/1
40		C anonymized	0/1
41		S anonymized	0/1
42		Connection type	-
43		P2P type	-
44		HTTP type	-

TCP Options Set			
C2S	**S2C**	**Short description**	**Unit**
65	88	RFC1323 ws	0/1
66	89	RFC1323 ts	0/1
67	90	window scale	-
68	91	SACK req	0/1
69	92	SACK sent	-
70	93	MSS	bytes
71	94	max seg size	bytes
72	95	min seg size	bytes
73	96	win max	bytes
74	97	win min	bytes
75	98	win zero	-
76	99	cwin max	bytes
77	100	cwin min	bytes
78	101	initial cwin	bytes
79	102	rtx RTO	-
80	103	rtx FR	-
81	104	reordering	-
82	105	net dup	-
83	106	unknown	-
84	107	flow control	-
85	108	unnece rtx RTO	-
86	109	unnece rtx FR	-
87	110	!= SYN seqno	0/1

Table 4.2: Flow Characteristics: Tstat Part Two

TCP End to End Set			
45	52	Average rtt	ms
46	53	rtt min	ms
47	54	rtt max	ms
48	55	Stdev rtt	ms
49	56	rtt count	-
50	57	ttl_min	-
51	58	ttl_max	-

TCP Layer 7 Set			
C2S	**S2C**	**Short description**	**Unit**
114		PSH-separated C2S	-
115		PSH-separated S2C	-
120		TLS Client ID reuse	-
121		TLS Client Last Handshake	ms
122		TLS Server Last Handshake	ms

This JSON structure includes information to make the connection between the TLS flow characteristics, taken by Tstat, and the fingerprint, taken by Suricata, Figure 4.1 and 4.2.

```
10.10.9.101 49182 44 1 43 4 4769 38 4987 1 218 0 1 0 103.68.223.153 6890 42 0 42 33 2019 8 2019 0 0 0 1 0 1616414124898.635986
1616414124907.957031 9.321000 0.276000 1.350000 9.272000 3.888000 0.100000 0.321000 1 1 0 0 8192 0 0 0.065939 0.043000 0.181000
0.042037 33 128 128 0.207663 0.047000 0.918000 0.349817 6 119 119 0 0 0 0 0 0 1 0 8 1 0 1460 1093 6 66560 8192 0 1162 69 166 0 0 0
0 2 0 0 0 0 1 0 8 1 1 1234 1234 6 17152 8192 0 1604 6 1604 0 0 0 0 0 0 0 0 0 0 --- 37 7 - - 0 0 0 2.000000 2.202000 2.348000
2.447000 333 1696 - - 0.0 0.0
```

Figure 4.1: Tstat Log Example


```
{"timestamp":"2021-03-22T11:15:03.199798+0000","flow_id":
2073309499222671,"in_iface":"tap0","event_type":"tls","src_ip":"10.10.9.101","src_port":
49182,"dest_ip":"103.68.223.153","dest_port":6890,"proto":"TCP","tls":{"subject":"C=FR, O=assylias.Inc,
CN=assylias","issuerdn":"C=FR, O=assylias.Inc, CN=assylias","serial":"1F:23:9D:BD","fingerprint":"d6:2e:06:53:11:df:fc:ec:ad:9f:-
8e:92:c3:16:aa:fb:60:19:39:4b","version":"TLS 1.2","notbefore":"2015-01-17T05:26:19","notafter":"2114-12-24T05:26:19","ja3":
{},"ja3s":{}}}
```


Figure 4.2: Suricata Log Example

In short, we can analyse and compare these characteristics: destination IP, destination port, source IP, source port. It is possible to put the two logs together and create a dictionary with the characteristics of each flow with its associated fingerprint.

4.3 C2 Traffic Description

The dataset used in this thesis was taken from the Malware-Traffic-Analysis.net website [35]. Where we collected information (pcaps) from various years (2013 to 2020). Table 4.3 presents some statistics about the data of the years collected.

Table 4.3: Dataset Statistics

Year	Nº of Malware TLS Flows	Nº of Bening TLS Flows	% of Bening TLS Flows
2013	0	165	100
2014	143	645	81.85
2015	1	445	99.78
2016	363	298	45.08
2017	499	2353	83.50
2018	6890	9391	42.32
2019	658	9623	93.60
2020	184	6260	97.14

Table 4.4 presents a count of the TLS flows that belong to each malware family in the dataset.

We know that the data is not correctly classified. The dataset is classified by comparing the fingerprint of each stream with a blacklist of fingerprints.

However, we know that the blacklist we use does not contain all the fingerprints associated with malware. Therefore, we conclude two things:

1. Data classified as malign is indeed malign (it contains the fingerprint on the blacklist);

Table 4.4: Malware Family Statistics

Family	Count
TrickBot C&C	5584
PandaZeuS C&C	3613
IcedID C&C	589
Gozi C&C	551
Gootkit C&C	329
Other	271
Dridex C&C	161
IcedId C&C	102
Shylock C&C	102

2. Data classified as benign could be either benign (it doesn't contains the fingerprint on the blacklist) or malign (it doesn't contains the fingerprint of a malignant flow on the blacklist, which shows that the list is incomplete).

The knowledge of the existence of misclassified data (malignant data classified as benign data) was acquired in the thesis [34]. Thus, we conclude that the detector will have to work with misclassified data, which will imply the existence of high false negatives.

4.4 Machine Learning Model

The model we use has an 88 feature input. The model is a dense neural network with the following layers:

- Input with 88 neurons (flow features)
- 1st layer with 256 neurons and Sigmoid Activation (hidden layer)
- 2nd layer with 128 neurons and Sigmoid Activation (hidden layer)
- 3rd layer with 32 neurons and Sigmoid Activation (hidden layer)
- Output with 1 neuron and Sigmoid Activation

We trained the model with an Adam optimisation algorithm [36] and used Categorical Crossentropy loss [37] to account for 2-categorical outputs.

4.5 Machine Learning Modes

We identify three modes of training machine learning: non-incremental offline training, incremental offline training and online training.

Non-incremental offline training is the classic method and likely the easiest to execute. Initially, we create two datasets, a training one and a test one to make the evaluation, Figure 4.3 [6].

Applying non-incremental offline training to the Malware Traffic Detector would be like splitting all known information randomly into two data sets and applying the concept: train, test and evaluate.

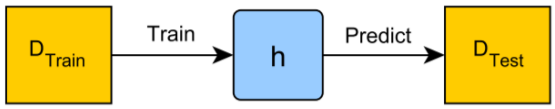


Figure 4.3: Non-incremental Offline Training [6]

Incremental offline training is partial the same as non-incremental training, the difference relates to the training method. This method trains incrementally, with training and evaluation blocks, Figure 4.4 [6]. This method is applied to our system in the way that at the beginning it is exactly the same as the previous method. The difference is that when the cycle ends, the method resorts to the previous model and continues to train the old model. Thus, the classifier model could be trained initially with all the data. Later, the training cycle are repeated (month by month) to give more knowledge to the existing model, which increase the classification ability and performance of the model.

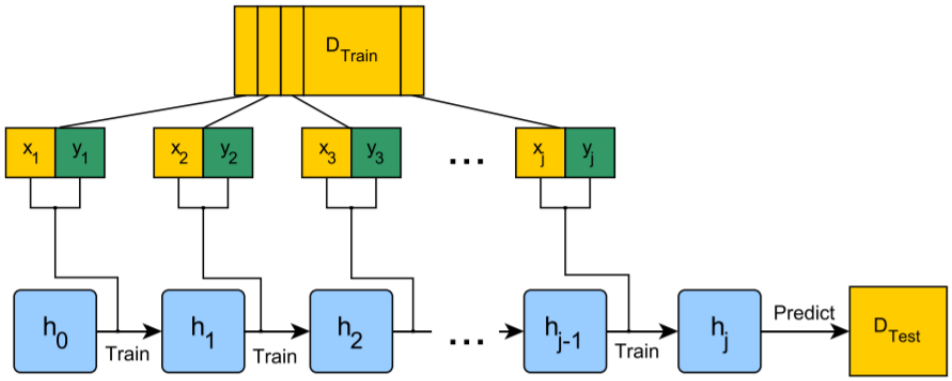


Figure 4.4: Incremental Offline Training [6]

Online training is much more challenging and different from the methods previously discussed, this method changes the way of model training. This method seeks maximum adaptation of the model to the information to be classified. This model uses one sample at a time to train, test and evaluate the model, continuing to evolve the model at each iteration, Figure 4.5 [6]. This method can be applied in our system, with a training, testing and evaluation cycle, leading to an increase in the model's knowledge with each sample.

The structure of the training models depends on the type of training: Non-Incremental Offline Training, Incremental Offline Training and Online Training. However, it was reported earlier the differences on the training cycle, for better clarification we will expose sections of the code that were made for the different types of training.

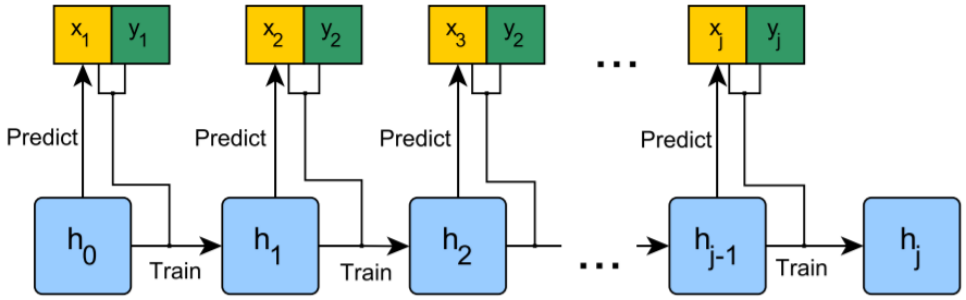


Figure 4.5: Online Training [6]

The batch size defines the number of samples to work with before updating the internal parameters of the model. The number of epochs defines the number of times the machine learning algorithm will run over the entire training dataset.

```
model = get_compiled_model()

model.fit(np.asarray(trainx), np.asarray(trainy), class_weight=class_weights,
    batch_size=32, epochs=10)

model.save(model_name)
```

Listing 4.1: Non-Incremental Offline Training

```
if available():
    model = load_model(model_name)
    model.summary()
else:
    model = get_compiled_model()

model.fit(np.asarray(trainx), np.asarraytrainy), class_weight=class_weights,
    batch_size=32, epochs=10)
model.save(model_name)
```

Listing 4.2: Incremental Offline Training

```
model = get_compiled_model()

for x in range(0,len(trainx)):
    model.fit(np.asarray(trainx)[x:x+1], np.asarray(trainy)[x:x+1],
    class_weight=class_weights, batch_size=32, epochs=50)
```

Listing 4.3: Online Training

Chapter 5

Offline Learning

In this chapter will we study the performance of our classification model.

Initially, we adopt the formation of a Baseline model classification to support the study of the system. Later, we will present a tool that allows us to manipulate the Classes Weight to improve the classification performance. Finally, we will establish a direct comparison between non-incremental and incremental training.

5.1 Best Classifier

Firstly, we intend to characterise the starting point for the study of the classifier. Thus, this will be a point of reference for the comparison with the studies developed throughout the research.

The first logical solution to implement the classifier was to use all existing information from the years 2013 to 2020. All further studies will use data from the MTA database[35] as a source of malware traffic. This first study also defines the baseline of training for the different studies.

5.1.1 Result

The base model performed as described in Tables 5.1 and 5.2. Note that all functions used on the calculations performed in the studies are described in Subsection 2.4.1.

Table 5.1: Confusion Matrix of the Best Classifier Experience

True label \ Prediction label	Benign traffic (TLS)	Malicious traffic (TLS)
Benign traffic (TLS)	4973	309
Malicious traffic (TLS)	79	2112

As it is possible to observe the model yielded very good results. The goal is to reduce the false negatives, since we are certain that they are positive. Observing the confusion matrix and the metrics table it is noticeable that there are many false positives. This phenomenon is due to the fact that there are some flows that are C2 and are known as benign, this problem will only be solved

Table 5.2: Metrics Table of the Best Classifier Experience

label	Precision	Recall	F1-score	Support
0.0	0.98	0.94	0.96	5282
1.0	0.87	0.96	0.92	2191
Accuracy			0.95	7473
Macro Avg	0.93	0.95	0.94	7473
Weighted Avg	0.95	0.95	0.95	7473

with the evolution of the knowledge of fingerprints of C2 flows. The false positives are flows that we will later analyse in order to realise that those flows are C2, in case they are, the fingerprints of the flows are added to the blacklist. The analysis work allows to improve the ground truth of the data used to train the classifier, as consequence the classifier performance will also improve.

5.2 Influence of the Class Weights

In this context, we have developed a tool that allows us to refine the detection of the classifier by performing decompensation on the class weights. Class weights are calculated with existing data and then adjusted relatively in percentage to the desired value.

This tool ends up being a filter that helps us to choose the traffic that has to be analysed to improve the knowledge of the blacklist, allowing to substantially decrease the amount of data to be processed.

The Listing 5.1 shows an use case: first we calculate the weights of the classes (ex: weight_for_0 and weight_for_1), secondly we define the weight of each class (ex: Y and X) and thirdly we calculate the final weight of each class according to the values defined.

```
counts = np.bincount(test_train_data[1][:].astype(int))
weight_for_0 = (counts[1]) / (counts[0] + counts[1])
weight_for_1 = (counts[0]) / (counts[0] + counts[1])
# Y -> 0's      X -> 1's
Y = 0.4
X = 0.6
class_weights = {0: ((weight_for_0*Y)/0.5), 1: ((weight_for_1*X)/0.5)}
```

Listing 5.1: Influence of the Classes Weight

Two comparison studies were carried out. One with a softer tuning, 1's at 60% and 0's at 40% versus 1's at 40% and 0's at 60%. The second tuning was more circumspect, 1's at 20% and 0's at 80% versus 1's at 80% and 0's at 20%. The comparison can be seen in tables 5.3, 5.4, 5.5 and 5.6, by order.

5.2.1 Result

We analysed the Tables 5.1 and 5.4 (Part: 1's at 80% and 0's at 20%), it was possible to see that the false negatives reduced from 79 to 32. However, the false positives increased from 309 to 439.

We also analysed the Tables 5.2 and 5.6 (Part: 1's at 80% and 0's at 20%), it was possible to see that in the 0's, Precision and Recall improved and F1-socore got worse. Meanwhile, it was possible to see that in the 1's, Recall improved and Precision and F1-socore got worse.

We conclude that it is possible to infer that it is feasible to fine tune the classifier to have a better performance in classifying 1's or 0's. In case of tool application, it would be preferable to optimise the classification of 1's.

By further analysing the data it is possible to infer that the model tuning brings more value to the classification of the desired characteristic, but deteriorates the classification of the opposite class. However, from our point of view, it is important to have the lowest number of false negatives, even compromising the false positives. This need is due to the existence of erroneous data in the training dataset (there are malign streams given as benign).

Meanwhile, this tool will help us to tune the model so that it does not classify (known) malign data as benign. At the same time, it allows pointing to traffic that has been classified as malign and is wrongly known as benign on the dataset, thus making it possible to identify traffic that needs to be analysed manually (correct the wrongly classification of the sample).

Table 5.3: Confusion Matrix of 1's at 60% and 0's at 40% vs 1's at 40% and 0's at 60%

1's at 60% and 0's at 40%		
Prediction label / True label	Benign traffic (TLS)	Malicious traffic (TLS)
Benign traffic (TLS)	4949	333
Malicious traffic (TLS)	80	2111
1's at 40% and 0's at 60%		
Prediction label / True label	Benign traffic (TLS)	Malicious traffic (TLS)
Benign traffic (TLS)	4965	317
Malicious traffic (TLS)	110	2081

Table 5.4: Confusion Matrix of 1's at 80% and 0's at 20% vs 1's at 20% and 0's at 80%

1's at 80% and 0's at 20%		
Prediction label / True label	Benign traffic (TLS)	Malicious traffic (TLS)
Benign traffic (TLS)	4843	439
Malicious traffic (TLS)	63	2128
1's at 20% and 0's at 80%		
Prediction label / True label	Benign traffic (TLS)	Malicious traffic (TLS)
Benign traffic (TLS)	5125	157
Malicious traffic (TLS)	289	1902

Table 5.5: Metrics Table of 1's at 60% and 0's at 40% vs 1's at 40% and 0's at 60%

1's at 60% and 0's at 40%				
label	Precision	Recall	F1-score	Support
0.0	0.98	0.94	0.96	5282
1.0	0.86	0.96	0.91	2191
Accuracy			0.94	7473
Macro Avg	0.92	0.95	0.94	7473
Weighted Avg	0.95	0.94	0.95	7473
1's at 40% and 0's at 60%				
label	Precision	Recall	F1-score	Support
0.0	0.98	0.94	0.96	5282
1.0	0.87	0.95	0.91	2191
Accuracy			0.94	7473
Macro Avg	0.92	0.94	0.93	7473
Weighted Avg	0.95	0.94	0.94	7473

Table 5.6: Metrics Table of 1's at 80% and 0's at 20% vs 1's at 20% and 0's at 80%

1's at 80% and 0's at 20%				
label	Precision	Recall	F1-score	Support
0.0	0.99	0.99	0.95	5282
1.0	0.83	0.97	0.89	2191
Accuracy			0.93	7473
Macro Avg	0.91	0.94	0.92	7473
Weighted Avg	0.94	0.93	0.93	7473
1's at 20% and 0's at 80%				
label	Precision	Recall	F1-score	Support
0.0	0.95	0.97	0.96	5282
1.0	0.92	0.87	0.90	2191
Accuracy			0.94	7473
Macro Avg	0.94	0.92	0.93	7473
Weighted Avg	0.94	0.94	0.94	7473

5.3 Non-Incremental Training

The present section studies the behaviour of the classifier over time. Firstly, began the study of the non-incremental training, described in Section 4.5, since it is the simplest to design, as it would also be the first step when evaluating the behaviour of the system over time.

5.3.1 Evolution Over the Years

We are interested in evaluating the system's evolution over time. The traffic in a network does not always have the same characteristics and may evolve over time.

Two studies were conducted that allow us to draw conclusions about the evolution of the system over time. In this case, we will use data from 2013 to 2020, MTA [35], excluding the years 2013 and 2015. In both excluded years, there was less than 5% of data compared to the other years, there wasn't enough data to train the model and their use in this experiment would misrepresent the experiment itself so we decided not to use them.

The first study (Study 1) was the training of one model of each year, with the data of the years in question. In the second study (Study 2), the model was trained for each year with all the data known up to that year.

To analyse in detail the models created from the two studies, two tests were created. One to verify the performance of each model with the data of each distinct year. In the second, the performance of each model was verified with all known data until the year of study. The performance of the models was evaluated in three characteristic – Precision, Recall and F1-score. Accuracy and Recall was calculated only on the classification of malign data since this is the objective of the model.

The data was grouped in sets of four graphs in order to make the comparison of the evolution, Figure 5.1.

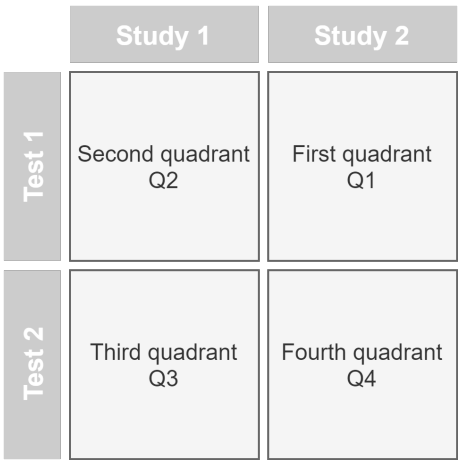


Figure 5.1: Graph decomposition legend (Non-Incremental Training)

5.3.1.1 Result

The comparison of the results was organised in 3 sets of figures, Precision 5.2, Recall 5.3 and f1-score 5.4.

Analysing Q2 (study 1, test 1) of the 3 sets it is possible to identify the discrepancy of information over the years. The best performance level occur at each year that was used to train the model (example: 2018 model has its maximum in year 3 (2018)). This behaviour indicates the existence of an evolution of the traffic morphology, since the values do not remain constant.

The change in the traffic morphology over time justifies the second training initiative. The comparison between the Q2 (study 1, test 1) of the 3 sets and the Q3 (study 2 test 1), allows us to verify that the use of all known data until the training date proves to be beneficial in comparison with the use of data relative only to that year. The values of the second study are substantially better; the models not only have a good performance for their relative year, but also a very good performance for the data of each previous year, which does not happen in the case of the first study.

Still, the comparison between the first quadrant (study 1 test 2) and the fourth quadrant (study 2 test 2), demonstrates the overwhelming advantage of using all known data up to the present year in model training. The second study proves that models trained with the all known data hold acceptable classification ability to classify all flows known to date. Whereas, the models of the first study perform poorly compared to those of the second study.

Key points:

- Traffic morphology evolves in time.
- Training with all the data known so far performs better compared to training with the data of the year in question, being better in the year being tested as well as in previous years.

The two studies performed allow to have a better idea of the change in the traffic morphology over time, while at the same time allowing to choose the best data to train the model, with the aim of increasing the performance of the classification system.

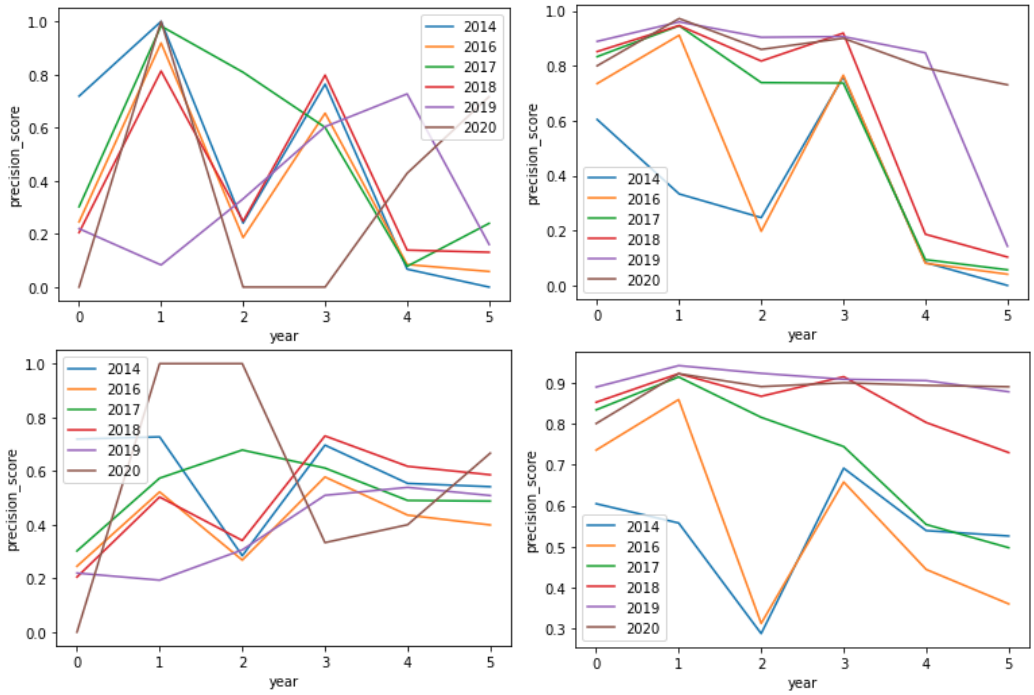


Figure 5.2: Comparison of Precision Score Non-Incremental Training: Evolution of the Classifier Over the Years

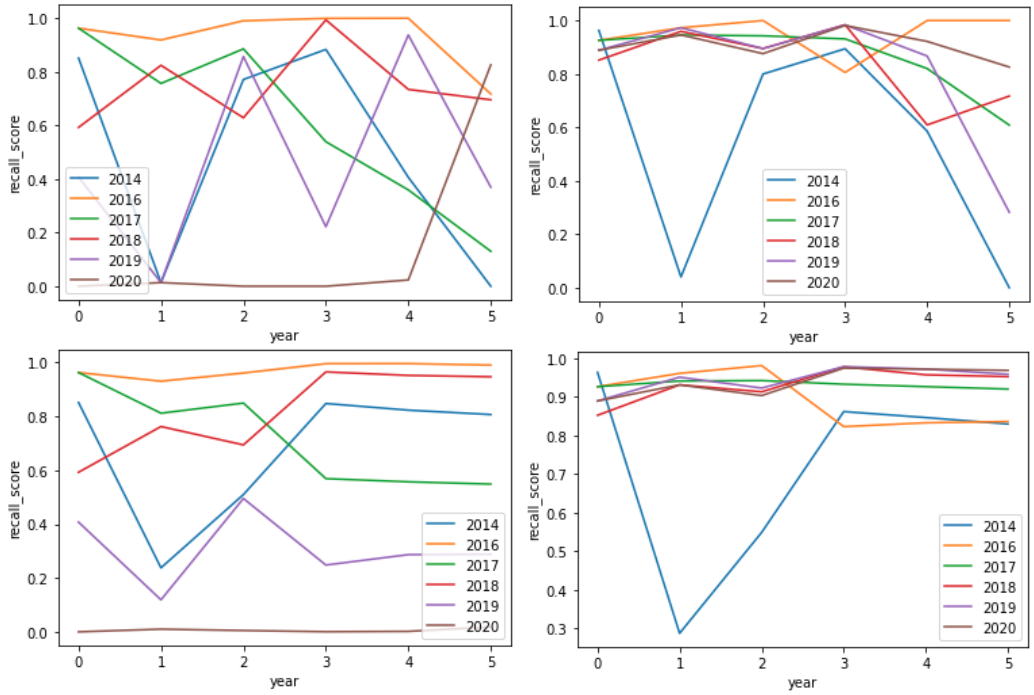


Figure 5.3: Comparison of Recall Score Non-Incremental Training: Evolution of the Classifier Over the Years

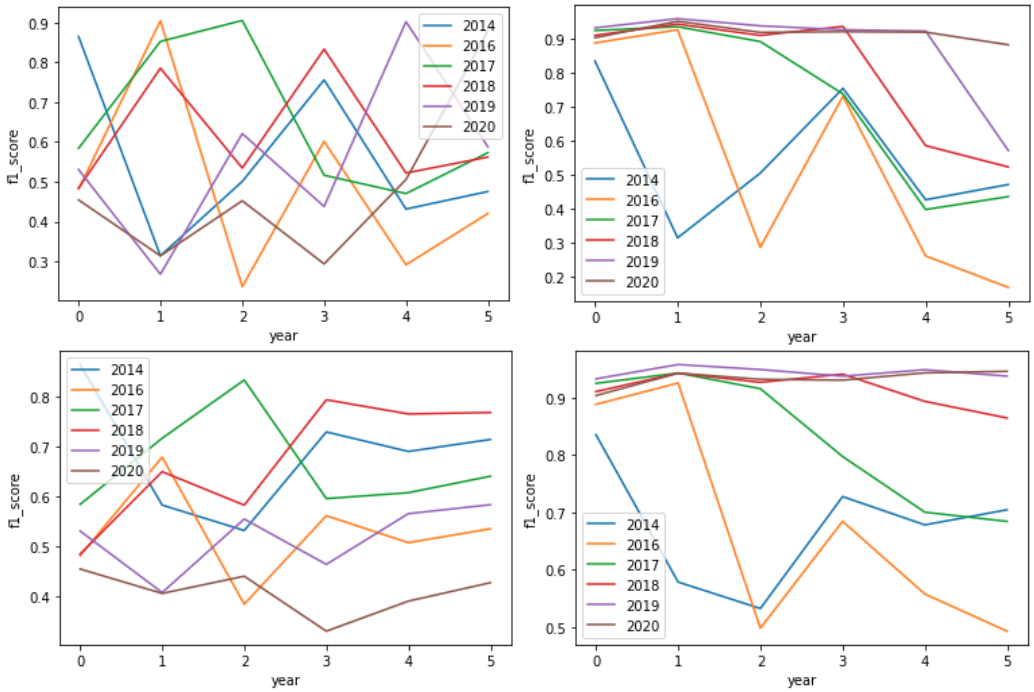


Figure 5.4: Comparison of F1 Score Non-Incremental Training: Evolution of the Classifier Over the Years

5.4 Incremental Training

This Section studies incremental training, described in section 4.5. It is necessary to verify that the use of this new training idea maintains the performance of the model in comparison with previous non-incremental training studies.

5.4.1 Evolution Over the Years

Incremental training comes almost as a necessity for us. Reflecting on the conclusions drawn from the previous section, it is noticeable that the traffic evolves over time, so there is a need to save all information that allows to create detectors with better performance. However, we are faced with a problem. The previous training system requires the saving of all existing information, as well as training with such data. We must have a large memory capacity, but above all, a large training capacity. Training the model non-incrementally would imply failure in the near future, since model training cannot be done in good time. To solve these two severe problems, incremental training was proposed.

Model training was carried out year by year incrementally, retraining the model of the previous year. The same two tests carried out in the two previous experiments were performed (Subsection 5.3.1), to allow direct comparison with them. In the first test, the performance of each model

was verified with data from each different year. In the second, the performance of each model was verified with all the known data until the years of study.

For a better comparison with the best study carried out in the previous section, we will dispose the information as illustrated below in Figure 5.5. The second study, the model was trained for each year with all the data known up to that year (non-incremental). The third study was carried out with incremental training, the training was carried out year by year (incrementally).

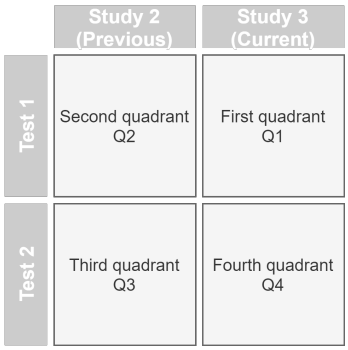


Figure 5.5: Graph decomposition legend (Incremental Training)

5.4.1.1 Result

The comparison of the results was organised in 3 sets of figures: Precision 5.6, Recall 5.7 and f1-score 5.8.

Comparing the two studies it is possible to verify that the incremental training is less efficient compared to the non-incremental training (all data). We can conclude that the incremental training begins to give less importance to the traffic of previous years, eventually starting to forget the older training and giving more importance to more recent data. Thus, we can resort to this method to induce this behaviour in the model using the incremental training method. However, it should be remembered that the data does not have the same weights since there are years with more data than others. It is possible to verify that in some years the performance values decay, like 2020 year. At the same time, sometimes the traffic of a certain year is not good enough for the incremental method, for example the year 2020, the model is very good in its year but is very bad for the previous years.

Thus, we can take away the following **points**:

- Incremental training models perform less effectively than non-incremental models with full data. However, the performance levels are acceptable.
- Incremental training models give more importance/weight to the most recent data, forgetting the data from older years. Tracks the evolution of traffic characteristics.

We can conclude from these results that it is not necessary to store all the information to train the model. This point allows optimising the training of the model in terms of storage as well

as time, since training with a temporal margin of data is much faster than training with the total dataset.

Meanwhile, there is a warning to us that some models from a certain year can easily forget the malware from previous years. To combat this problem, we can save the various stages of the models and use the models in parallel to get a more accurate classification.

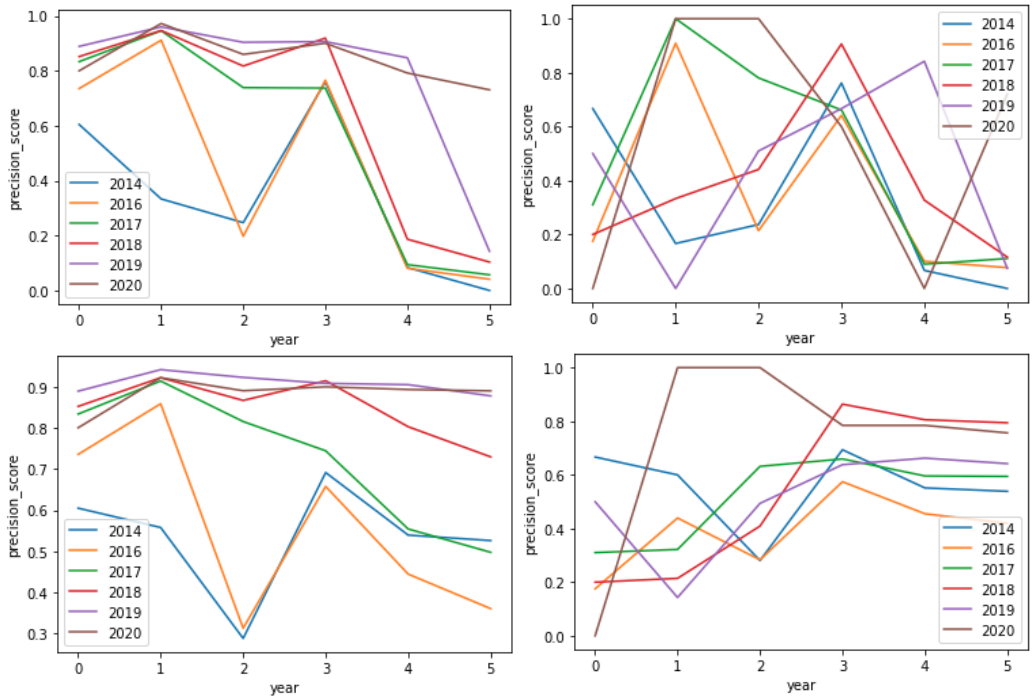


Figure 5.6: Comparison of Precision Score Incremental Training: Evolution of the Classifier Over the Years

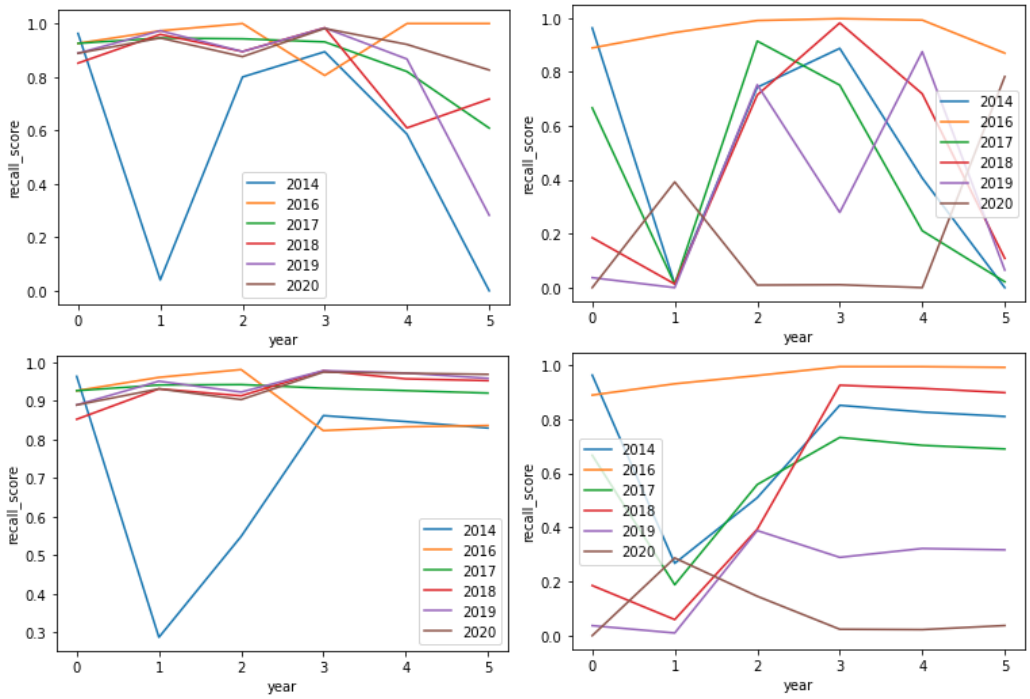


Figure 5.7: Comparison of Recall Score Incremental Training: Evolution of the Classifier Over the Years

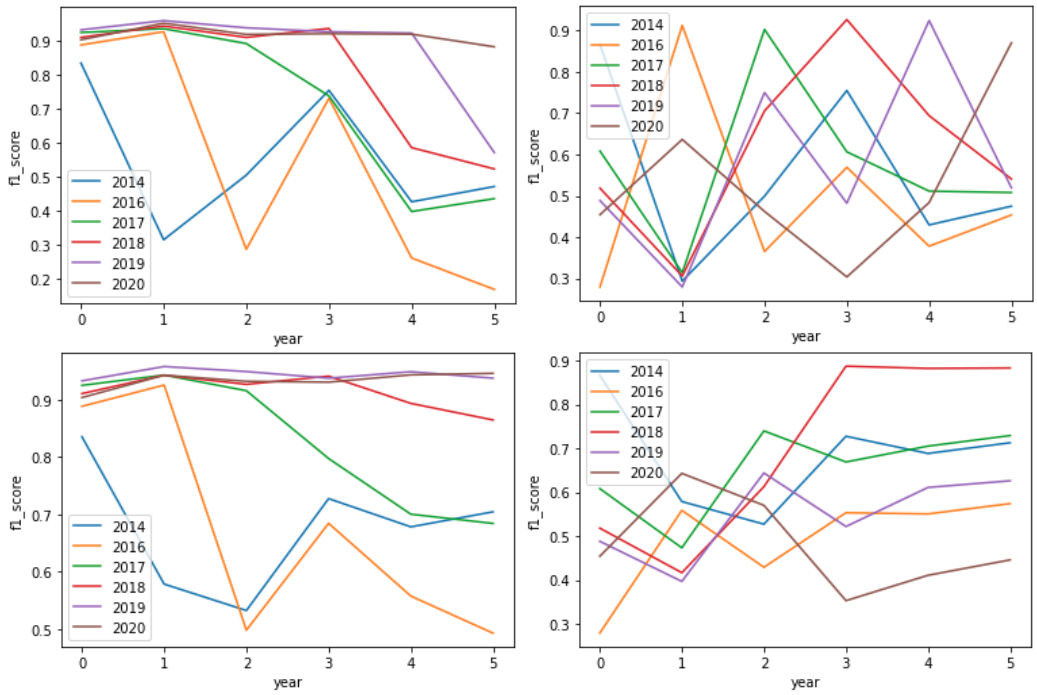


Figure 5.8: Comparison of F1 Score Incremental Training: Evolution of the Classifier Over the Years

5.5 Conclusions

The results of the studies show the evolution of the traffic characteristics over time and the verification of the performance of the classification model with different training approaches.

Section 3.3.3, we improved the performance of the base model. Section 5.2, demonstrated the possibility of tuning the detector classification system in order to improve the classification performance of a particular class.

Section 5.3, we did a deeper study of the evolution of malware in time. We proved that the characteristics of traffic change over time, as it is also proved a possibility to adapt the models to follow the evolution of traffic. However, we concluded that there is a need to preserve and use all the existing data to improve the model's classification performance as much as possible.

In section 5.4, a study was developed with the purpose of eliminating the need to use all data in the formation of models. The comparison of the incremental training approach with the non-incremental one solves the storage/total training problem, but presents a lower yet acceptable performance comparing to the models that use all data in a single training.

Overall, the studies allowed reaching the objectives required for the detection system (ML classifier), designed in Chapter 3 to work properly. Nevertheless, it also adds knowledge in the area of malware detection, providing proof of operation of new concepts of training malware detectors. This knowledge can help save resources and optimise detection systems.

Chapter 6

Online Learning

The previous chapters explain and demonstrate how the intrusion detection system works, as well as proving that the options taken are in line with what was thought, the evolution of traffic characteristics in time and the evolution of the detection system in relation to changes in traffic over time.

In this chapter, we begin to explore the development of online training, aiming to integrate it into the intrusion detection system produced in this thesis.

The exploration of the model in the online area, aims to improve the overall performance of the system. An online training system as explained in Chapter 2 and 4 has a higher capacity to learn faster than offline training [6]. Therefore, the model will adapt more easily to traffic changes and increases the probability of finding zero-day attacks.

In contrast, the introduction of the online system reduces the complexity of the detection system in general, since it would no longer be necessary to store all the information. We could simply save false positives for later analysis, so we could continue to improve our blacklist. So, this solution would easily store lower amounts of data.

This chapter is divided in two main sections, the first one will prove the possibility of achieving the desired solution 6.1 and the second one will apply the concept to real data 6.2.

6.1 Impact of Different Sequences of Labels

The application of online training in the detection system requires the use of a training library. However, the library used in the design of the offline training did not have any reference in the documentation about the possibility of online training. Nevertheless, doing some research we came to the conclusion that it might be possible to apply the online method with the Keras library.

However, in this Section 6.1, we had to perform a set of studies to proof the online training functionality, as well as verify some limitations of the online approach (like label sequence - 0's and 1's).

6.1.1 Synthetic Dataset

The experimentation of the online method has made under controlled conditions to ensure that our system will always behave in the same way. With this in mind, we designed a dataset with synthetic data.

The synthetic data was created to look as similar as possible to the real data, having 88 characteristics **nsamples**. The **nsamples** is the number of samples that each class will have. The synthetic data is random, generated by multivariate Gaussians with the average of each dimension in a vector **loc** and with diagonal covariance matrix, which means the variables are uncorrelated. Creating two different vectors (**loc0** and **loc1**) allows us to create the two classes of 0's and 1's. The **nsamples** is the number of samples that each class will have. The creation of the datasets is shown in Listing 6.1.

```
import numpy as np

ndim = 88
nsamples = 50000
loc0 = np.random.uniform(0,1,ndim)
scale = np.ndarray.tolist(np.eye(ndim))
values = np.random.multivariate_normal(loc0, scale, nsamples)
values0 = np.clip(values, 0, 1)
values0_y = [0] * nsamples

loc1 = np.random.uniform(0,1,ndim)
scale = np.ndarray.tolist(np.eye(ndim))
values = np.random.multivariate_normal(loc1, scale, nsamples)
values1 = np.clip(values, 0, 1)
values1_y = [1] * nsamples
```

Listing 6.1: Creation of Synthetic Dataset

6.1.2 Comparison of Offline Training with Online Training (synthetic data)

The comparison between the two trainings was made by using three studies. However, it should be noted that the structure of the models remains the same as the one assigned in Chapter 4, it differs in the training method and number of epoch to 10. As it was explained in Section 4.5, offline training feeds the **FIT()** function with all the training data, while the online training feeds the **FIT()** function sample by sample.

Initially, the offline model was designed to understand if the offline model can learn with a synthetic dataset, as well as to serve as a means of comparison with the online version. Later, online training was carried out, one with the unmixed dataset classification, example [00...0011...11], and another with the mixed one, example [0100011101010010110] having 0's and 1's classification randomly alternated.

Two metrics were computed in the three studies the confusion matrix and the metrics table. Although, three graphs were created to analyse accuracy, recall and F1-score for the two studies

using online training. There is a need to mention that the confusion matrix and the metrics table created for the online training are an average indication of the final state in which the training ended. In addition, for a more detailed analysis of the metrics, it will be necessary to analyse the evolution graphs over time.

Table 6.1: Confusion Matrix Comparison Offline Vs Online Training

Offline Training		
Prediction label / True label	Benign traffic (TLS)	Malicious traffic (TLS)
Benign traffic (TLS)	9656	344
Malicious traffic (TLS)	268	9732
Online Training Non-Shuffled		
Prediction label / True label	Benign traffic (TLS)	Malicious traffic (TLS)
Benign traffic (TLS)	0	2000
Malicious traffic (TLS)	0	2000
Online Training Shuffled		
Prediction label / True label	Benign traffic (TLS)	Malicious traffic (TLS)
Benign traffic (TLS)	1956	44
Malicious traffic (TLS)	244	1756

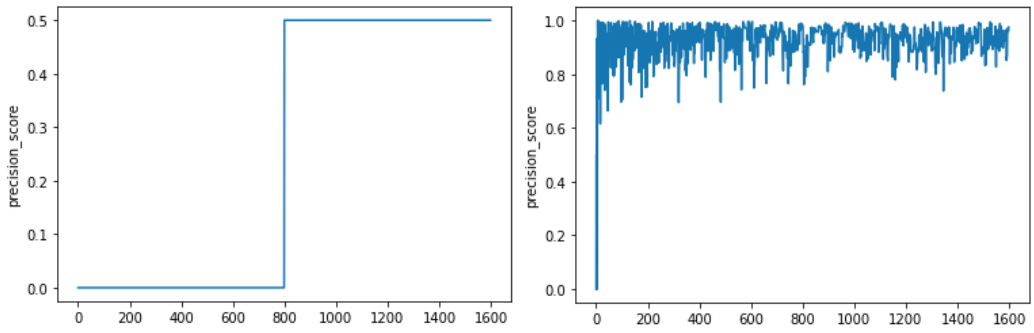


Figure 6.1: Precision Score Comparison of Non-shuffled Vs Shuffled Online Training

Analysing the Tables 6.1 and 6.2, it is possible to observe that the model with offline training obtained a good result, exactly as expected. However, the analysis of the online training, we obtain two completely opposite cases. The unmixed training fails completely, but the mixed training gets promising results. A a more detailed analysis can be done by looking at the comparison Figures 6.1, 6.2 and 6.3. It should be noted that the X axis represents the evolution over cycles, so it allows to analyse and compare the different evolutions over cycles for each training. Analysing the graphs over cycles we verify the same as in the tables the unmixed training misses its target completely and the mixed training obtains promising results.

This behaviour raises a question - "*Does the method used allow online training?*". Under certain conditions we think so, but the distribution of zeros and ones seems to influence the training

Table 6.2: Metrics Table Comparison Offline Vs Online Training

Offline Training				
label	Precision	Recall	F1-score	Support
0.0	0.97	0.97	0.97	10000
1.0	0.97	0.97	0.97	10000
Accuracy			0.97	20000
Macro Avg	0.97	0.97	0.97	20000
Weighted Avg	0.97	0.97	0.97	20000
Online Training Non-Shuffle				
label	Precision	Recall	F1-score	Support
0.0	0.00	0.00	0.00	2000
1.0	0.50	1.00	0.67	2000
Accuracy			0.50	4000
Macro Avg	0.25	0.50	0.33	4000
Weighted Avg	0.25	0.50	0.33	4000
Online Training Shuffle				
label	Precision	Recall	F1-score	Support
0.0	0.89	0.98	0.93	2000
1.0	0.98	0.88	0.92	2000
Accuracy			0.93	4000
Macro Avg	0.93	0.93	0.93	4000
Weighted Avg	0.93	0.93	0.93	4000

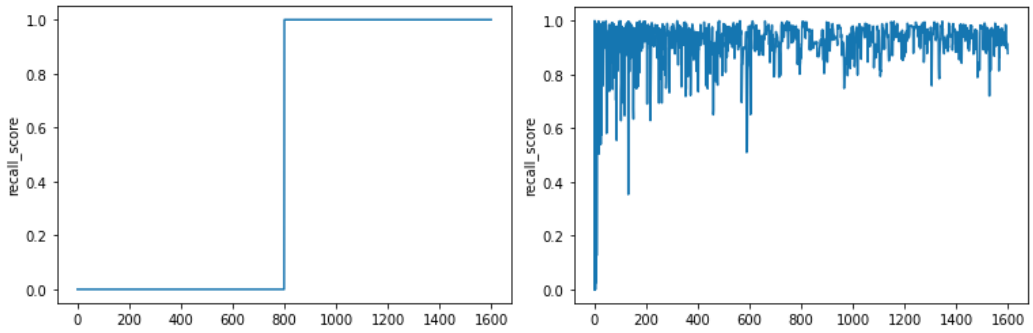


Figure 6.2: Recall Score Comparison of Non-shuffled Vs Shuffled Online Training

capacity of the method used. We elaborate further on this in subsection 6.1.3.

6.1.3 0's & 1's

The discovery of the limitation in the training sequence implies a deep analysis of this obstacle, since the sequence of the classification with real numbers is not controlled. To solve this issue, we decided to create a set of tests that explore the training capabilities of the online method.

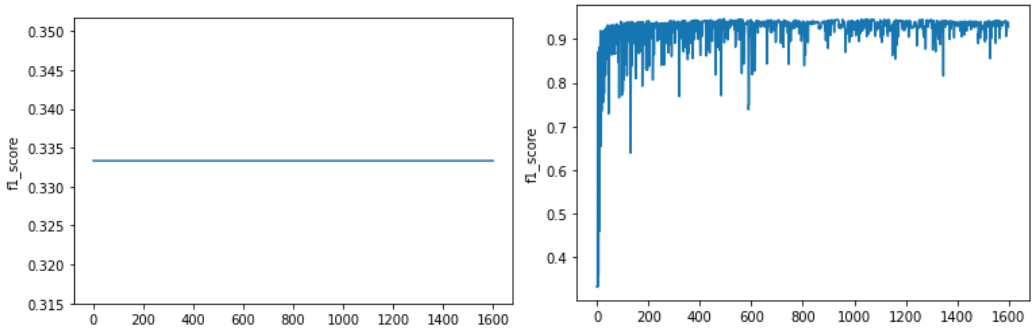


Figure 6.3: F1 Score Comparison of Non-shuffled Vs Shuffled Online Training

Initially, we started to make tests with different sequences of 0's and 1's, to find the 0's and 1's limitations. We explored the following combinations of 0's/1's represented in Table 6.3.

Table 6.3: Sequences of 0's and 1's

	Study 1	Study 2	Study 3	Study 4
0's	32	128	128	128
1's	32	128	32	32
Epoch	10	10	10	1

The graphs were organised in this manner, Figure 6.4, to be able to make a direct comparison of the results. We chose to use the same metrics as before, Precision, Recall and F1-score.

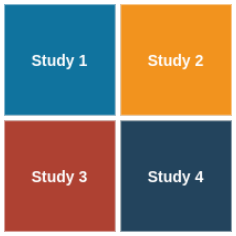


Figure 6.4: Graph decomposition legend

Looking at the graphs it is noticeable that the first study was a good performance and provides the basis for finding the limits of the approach. Then, the second study has many defects, and the performance varies within an acceptable range. In the third study, it is clear that we have reached the limit of the acceptable performance.

After several attempts, we discovered the limit of the method used, as was the case in study three. We changed the approach to the problem and started to try different cases. We came to the conclusion that reducing the number of epochs per data allows to improve the training performance when the method operates at its limit. As it is possible to observe in fourth study, more benign flows were chosen than malign ones, which is normal for a common network. Observing the fourth graph, it is perfectly possible to operate an online training up to the extreme patterns of the fourth study.

Key point:

- Online training is approved under specific conditions.

By performing this verification, we can also point out that this type of training can be applied both to a malware setup created by us (class distribution - 50%/50% - 0's/1's), and also to real data (class distribution - 80%/20% - 0's/1's).

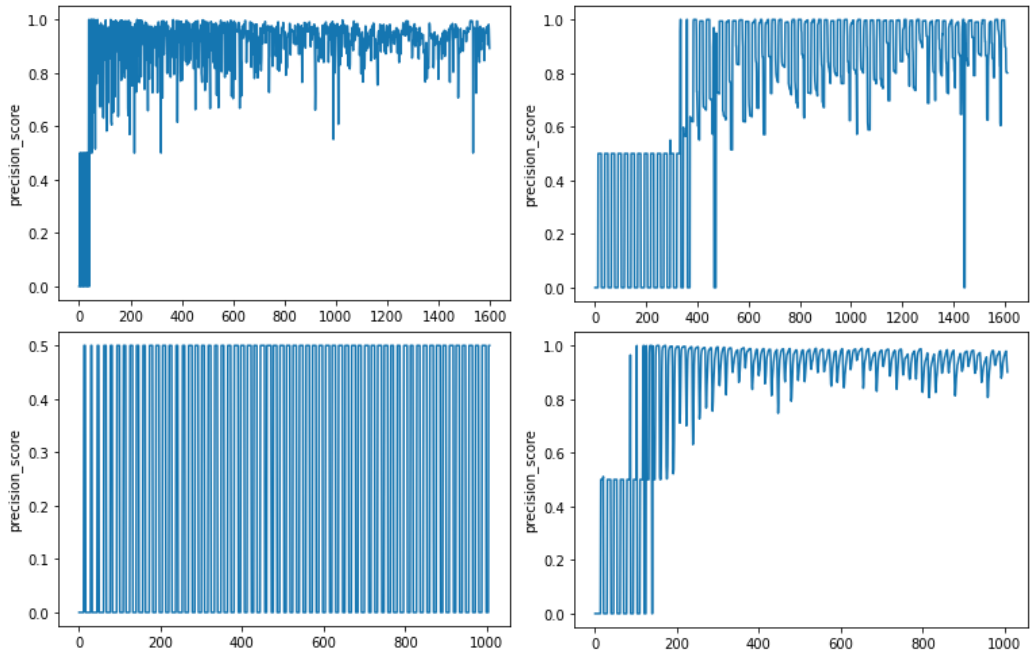


Figure 6.5: Comparison of Recall Score Incremental Training

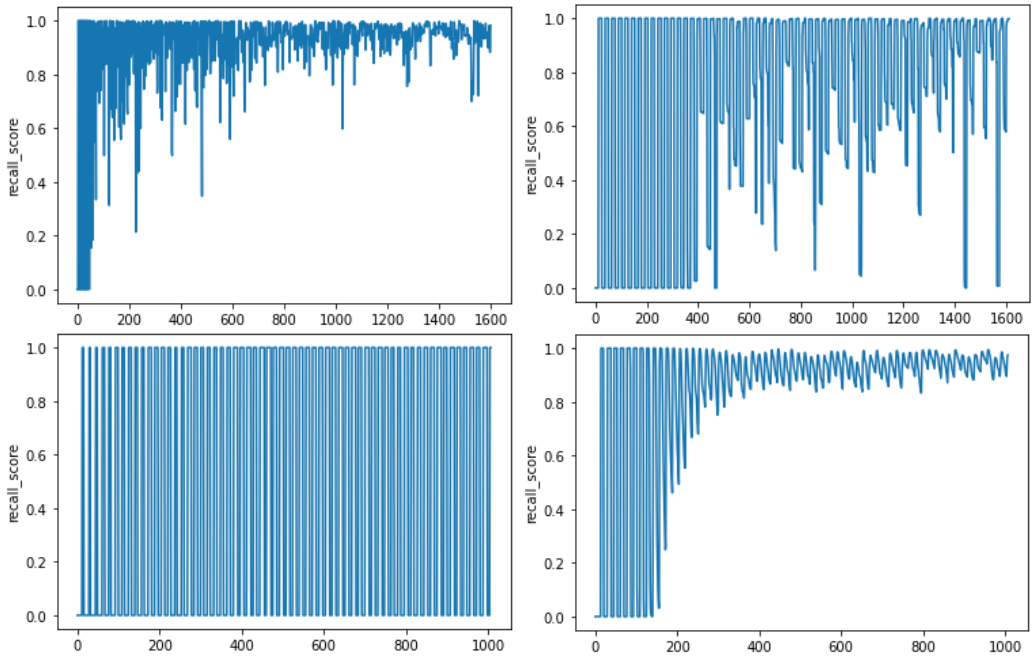


Figure 6.6: Comparison of Precision Score Incremental Training

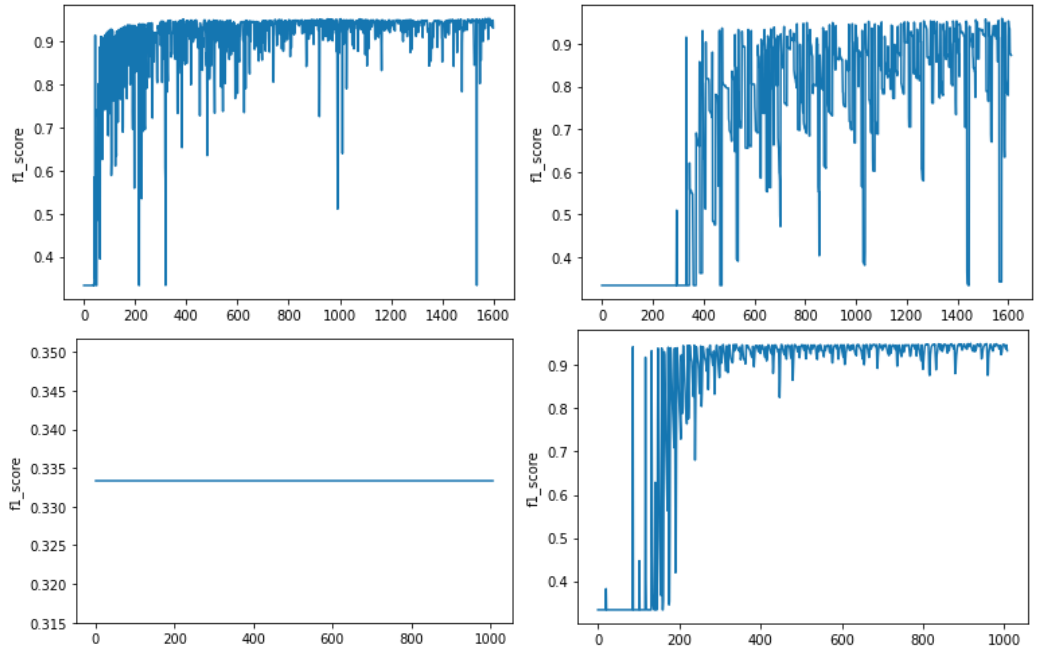


Figure 6.7: Comparison of F1 Score Incremental Training

6.1.4 Conclusions

Studies have shown that online training is possible. However, it should be remembered that it is feasible within the operating margins extrapolated in the verification tests.

6.2 Application to the C2 Dataset

Application with real data is more important than synthetic data. Therefore, this topic will be addressed in the next subsections.

6.2.1 Operation with Real Data

Firstly, we want to demonstrate that online learning works properly with real data, so we designed two tests in ideal and equal conditions to compare the performance of offline non-incremental training with online training.

We randomly choose 10k data from the total data set and we always taking care that the sequence of 0's and 1's does not exceed the online learning limit. All training characteristics were equal, even the number of epochs to 10. The only differences on online learning are the bash_size=1 and the number of samples passed at each iteration to fit() is one sample.

The first study refers to offline training, which will serve as the basis of performance comparison. The second study refers to the online training, which aims to demonstrate how the approach works with real data.

The main means of comparison are the confusion matrix and the performance table (Precision, Recall and F1-score). Again, it is important to remind that these indicators, in the case of online training, simply refer to the last state that the model finished the training. To analyse the evolution and the different states of online training over time, three graphs will be shown, Precision, Recall and F1-score.

Table 6.4: Confusion Matrix Comparison Offline Vs Online Training with Real Data

Offline Training		
Prediction label / True label	Benign traffic (TLS)	Malicious traffic (TLS)
Benign traffic (TLS)	3875	1407
Malicious traffic (TLS)	226	1965
Online Training		
Prediction label / True label	Benign traffic (TLS)	Malicious traffic (TLS)
Benign traffic (TLS)	3847	1435
Malicious traffic (TLS)	272	1919

Comparing the two studies we notice that online training with real information is completely feasible in a controlled environment. Analysing the performances of the two trainings, it can be seen that the performance of online training is identical to offline training with the same training characteristics. It is possible to observe that the offline training had a lower performance than the base model presented in section 5.1. The problem is due to the choice of training characteristics, such as the number of epochs (Keras library implies the use of a low number of epochs to work within the operating limit). We used a low epoch number to operate within the limitations of the online approach, so essentially, we benefited online training in this comparison. However, it

Table 6.5: Metrics Table Comparison Offline Vs Online Training with Real Data

Offline Training				
label	Precision	Recall	F1-score	Support
0.0	0.94	0.73	0.83	5282
1.0	0.58	0.90	0.71	2191
Accuracy			0.78	7473
Macro Avg	0.76	0.82	0.77	7473
Weighted Avg	0.84	0.78	0.79	7473
Online Training				
label	Precision	Recall	F1-score	Support
0.0	0.93	0.73	0.82	5282
1.0	0.57	0.88	0.69	2191
Accuracy			0.77	7473
Macro Avg	0.75	0.80	0.76	7473
Weighted Avg	0.83	0.77	0.78	7473

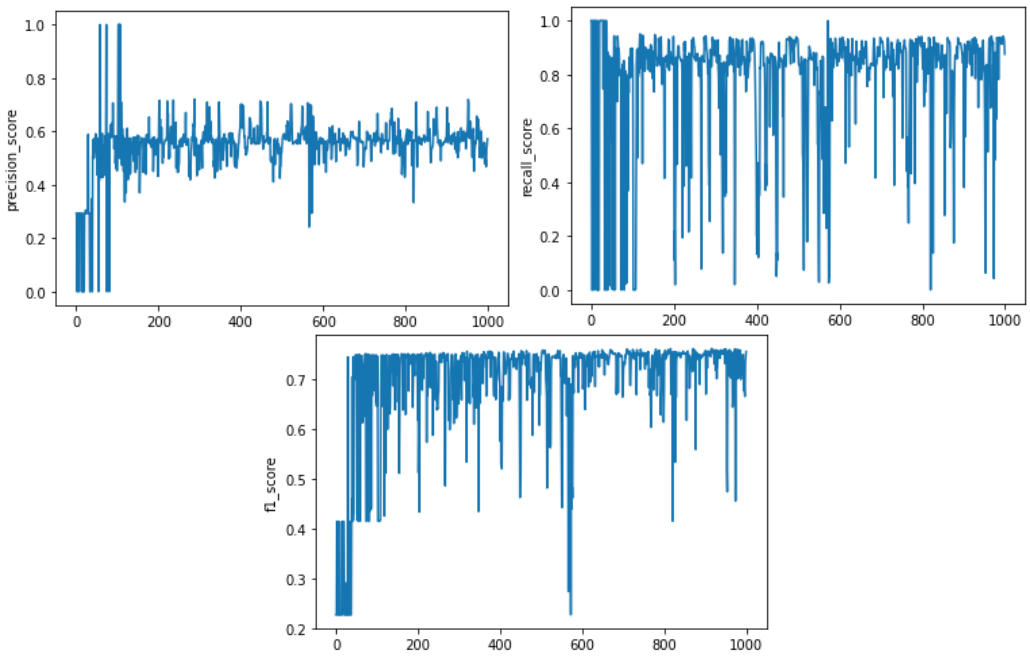


Figure 6.8: Online Training - 10k Real Shuffled Data

was an option taken with the intention of being able to produce a fairer means of comparison, since having the same training characteristics and the same data allows us to have a better direct comparison of the trainings.

Analysing the Figure 6.8 it is possible to infer that most of the evolution over time the model has remained within the operating margins of online learning, however there are some points where there is a loss of performance.

Overall, we demonstrate that online training can be applied to real data in a controlled environment.

Therefore, a question was raised - "Does it work if we use real data in its natural order?". In the two previous studies, 10k data were chosen randomly taking into account the sequences of 1's and 0's. However, the real data does not have controlled sequences hence the third study was born.

The third study comes to demonstrate what was determined with synthetic data, that it is not possible to do online learning with the normal sequences of real data. Therefore, the study was conducted with the same characteristics as the previous studies, the difference was only the training data, where we chose the year 2018 for the test without using data shuffle.

Figure 6.9 presents the analysis of the evolution results with Precision, Recall and F1-score graphs.

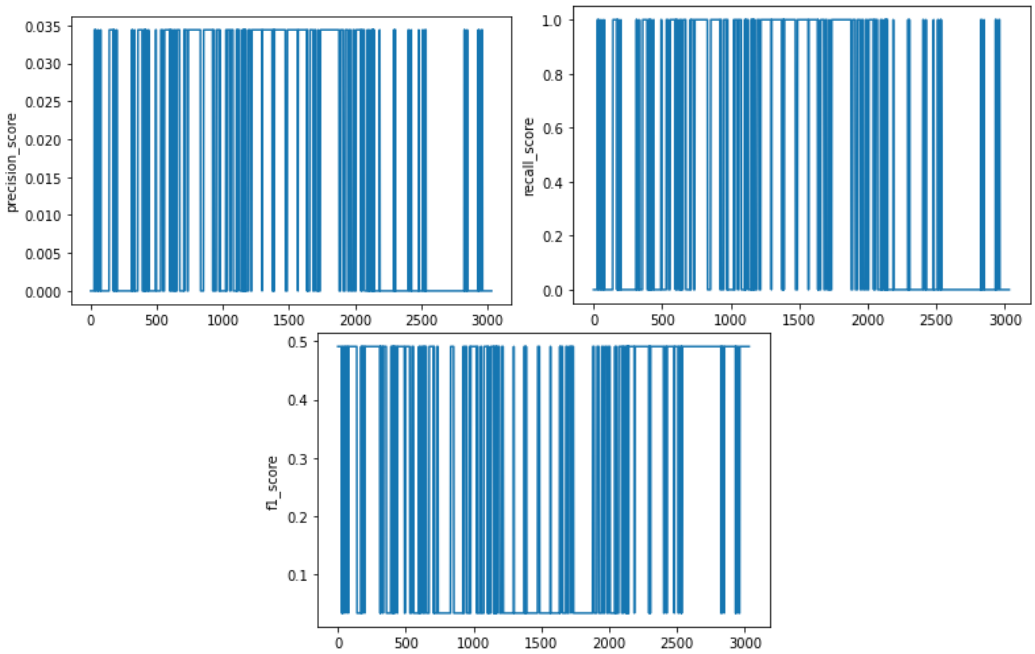


Figure 6.9: Online Training - Year 2018 Real No-shuffled Data

Analysing the Figure 6.9, it is noticeable that online training with data in its natural sequence (No-shuffled) does not have an acceptable performance. This conclusion comes in agreement with what was concluded with the synthetic data. Therefore, we must move on to the next step to determine the limit of our approach with real data, which we discuss in the next Subsection 6.2.2.

6.2.2 0's & 1's Tuning

Once we prove that the online training technique can be applied to real data in a controlled environment, the next step is to fine tune the training characteristics. The distribution of 0's and 1's proved to be the main influence on training performance. However, it should be remembered that a low epoch number (epoch=1) is important for performance too. This sub-chapter will look at

the influence of the sequences of 0's and 1's on real data. These studies will give us the know-how, allowing to fine-tune the training model according to the preferences, as well as to the data available. The studies will have three associated graphs, which have associated metrics (Precision, Recall and F1-score).

The studies were performed with real data, however each study used different distributions of 0's and 1's. The training datasets were created according to the desired distribution, according to the chronological order of the flows, X flows of 0's and Y flows of 1's were grouped, as shown in the Listing 6.2.2.

```
// ex: 128/32 -> 0s Vs 1s
X = 128
Y = 32

treinox = []
treinoy = []
y = aux
for x in range(0,aux-1,X):
    for aux1 in test_train_data[0][x:x+X]:
        treinox.append(aux1)
    for aux1 in test_train_data[0][y:y+Y]:
        treinox.append(aux1)
    for aux1 in test_train_data[1][x:x+X]:
        treinoy.append(int(aux1))
    for aux1 in test_train_data[1][y:y+Y]:
        treinoy.append(int(aux1))
    y += Y
```

Listing 6.2: Creation of Real Dataset

Table 6.6 follow the tests that were performed.

Table 6.6: Table of Studies of the Distribution of 0's and 1's

Study	1	2	3	4	5	6
Distribution (0/1)	128/32	64/32	32/32	32/16	32/8	32/4

Initially, we followed the path used in synthetic data. It started with higher sequences as is the case of study one (A.8) and two (A.9). Analysing the results, it is possible to observe that the results were not acceptable since they take a long time to converge and suffered several breaks during the operation time. From these two studies we conclude that the 128/32 and 64/32 ratios are not possible when using real data.

The next step was to reduce the number of each class, following the line shown in the synthetic data. Where we created the study three A.10, four A.11 and five A.12. Analysing the results, it is possible to observe that there is always a period of adaptation on the initial part of the model (convergence of the model). However, when analysing the three models it is possible to observe the improvement of performance with the reduction of the number of each class (0's and 1's

distribution). This feature is in accordance with what was concluded in the study with synthetic data. In addition, we can affirm that studies three, four and five have acceptable performances since most of the time they have performance values in the operating margin. However, taking into account the sharp decreases in performance, it will be necessary to find countermeasures to alleviate these problems. From these three studies we conclude that the ratios 32/32, 32/16 and 32/8 can be used in online training, since they have a good performance, but it should be noted that they have some sudden decreases in performance.

Finally, the best duality between performance and the sequencing of classes was sought. From this search we create the study six (A.13). The study is composed by a sequence of 32 (0's) and 4 (1's). Analysing the results of the study, we can affirm that it presents a high performance. In turn, it no longer presents the behaviour of abrupt decrease of performance, so we can conclude that study six has a high performance. In turn, it no longer presents the behaviour of abrupt decrease of performance, so we can state that the decreasing the number of each class allows eliminating this problem.

However, the possibility of using online training with real data was proven, the studies carried out allowed us to figure out the boundaries of operation of online training. Table 6.7 and Figure 6.10, allows us to map the limitation of training in relation to the sequence of 0's and 1's. Analysing the Figure 6.10, allows us to group the studies into 3 groups in relation to their performance evaluation, where they are represented and classified in Table 6.7. Note that the evaluation goes from A to D, where A is the best performance evaluation and D is the worst performance evaluation.

Table 6.7: Table of Studies of the Distribution and Evaluation of 0's and 1's

Study	1	2	3	4	5	6
Distribution (0/1)	128/32	64/32	32/32	32/16	32/8	32/4
Evaluation	D	D	B	B	B	A

In conclusion, the sequences of 0's and 1's used to obtain acceptable performances with real data are much inferior to the synthetic data. This conclusion raises another problem that is which are the sequences in the real data, in these studies we made a dataset with the desired sequences. However the real data does not have this behaviour and this subject will be addressed in the next Subsection 6.2.3.

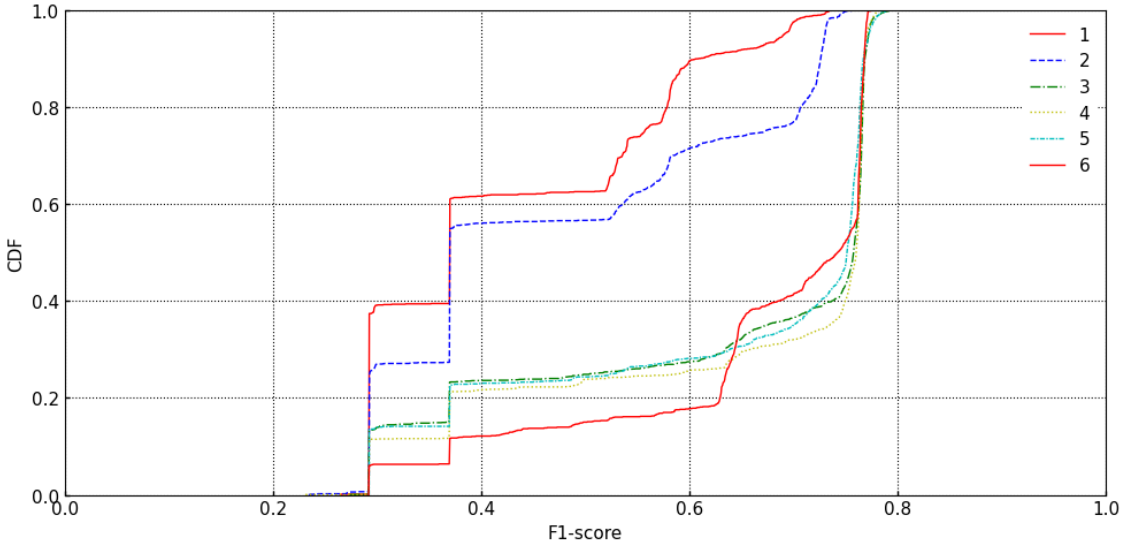


Figure 6.10: CDF F1-score Studies

6.2.3 Class Distribution

The distribution of the data in the natural sequence of the live data is unknown. This problem has to be solved so that the we can choose our training options, since the success of online training is dependent on the class sequences.

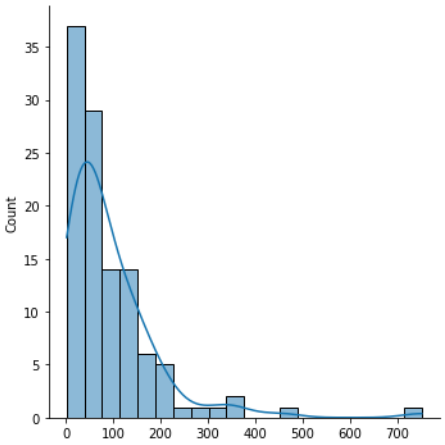
A study has been constructed which gives an insight into the distribution of classes over time and from this we can draw conclusions about the possibility of online training, Figure 6.11a. On the Figure the X axis counts the number of times that a sequence is found and the Y axis the number of the sequence (ex: 10 -> 1's/0's).

The previous study, Figure 6.11a, are about the sequences of the two classes, however it is necessary to make a study for each class to be able to have a perspective of the ratio between the two classes and it is another important characteristic in the online training, for that it was created the study of the sequences of the 0's 6.11b and the 1's 6.11c.

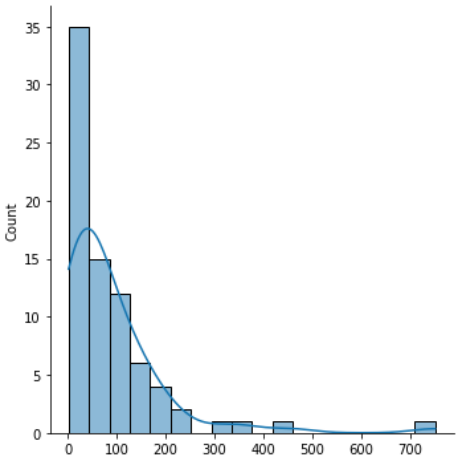
Analysing the distributions of sequences it is possible to observe that they exceed the operation limits for online training (defined on Subsection 6.2.2), but they are close to the operation level. Therefore, it is necessary to find methods to minimise this problem. One solution would be to simply discard certain flows for training in order to obtain the required sequence. In short, the online training would not be done in every iteration, but when it is possible to maintain the ratio and the desired sequence of the training classes. However, this problem needs to be addressed and discussed in future work.

However, not all years have the same sequences. For this reason we made a detailed analysis of each year regarding the sequence of 0's and 1's, we exposed all years in the appendix, in (A.4). Therefore, here we will give an example of a good year 6.12 and a bad year 6.13. In order to

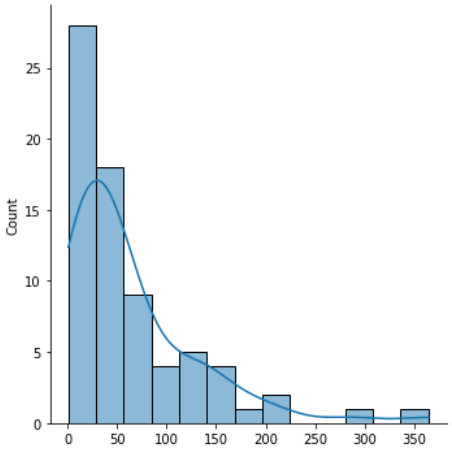
reinforce the idea given earlier, that online training is not always able to operate naturally. Note that good year is inside the operation limits for online training and bad year exceed the operation limits for online training (defined on Subsection 6.2.2).



(a) Class Distribution

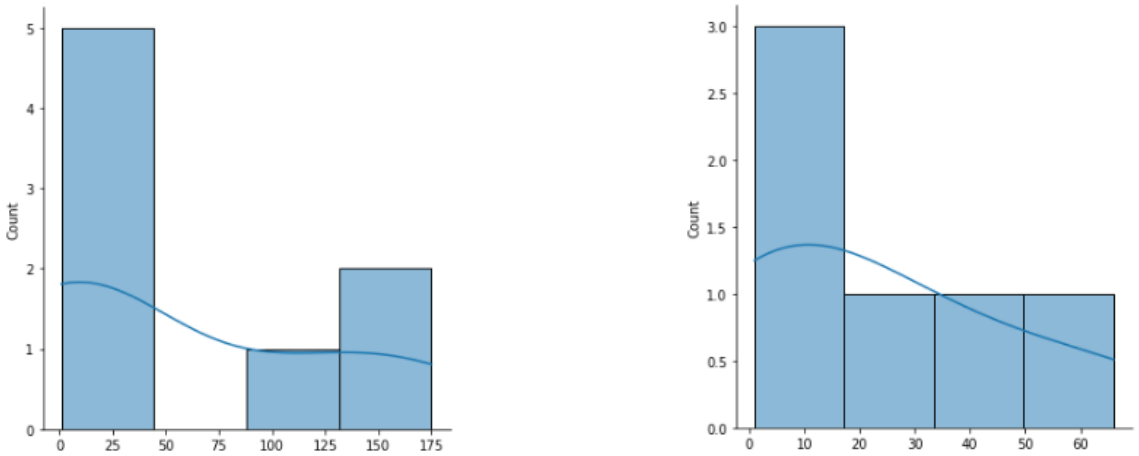


(b) 0's Distribution Study



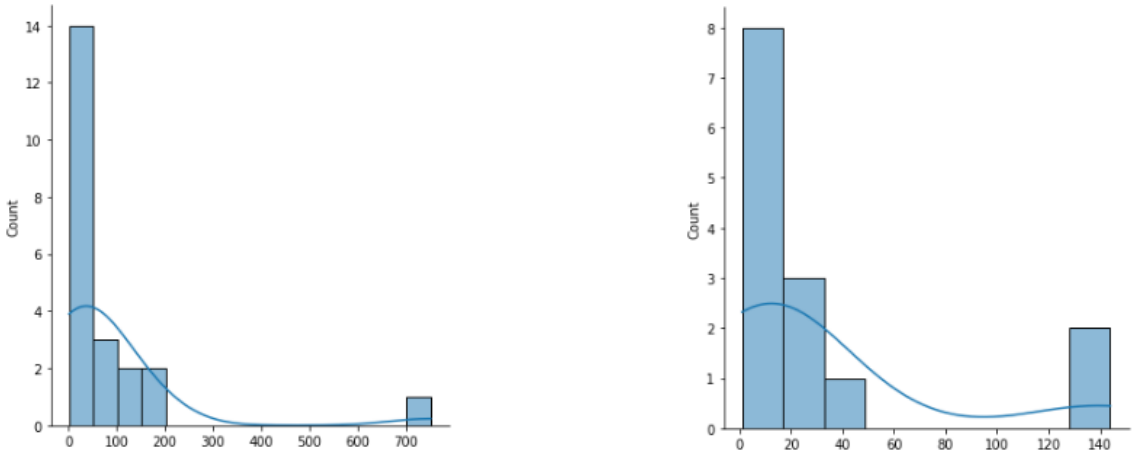
(c) 1's Distribution Study

Figure 6.11: Class Distribution Study



(a) 0's good year (b) 1's good year

Figure 6.12: Good year



(a) 0's bad year (b) 1's bad year

Figure 6.13: Bad year

6.3 Conclusions

Throughout the chapter, lessons have been learned that will help analysts exploit detection systems supported by online training.

We demonstrate that it is possible to use the Keras library for online training, as well as to overcome the limits of our approach. However, it was found that the limits imposed by the synthetic data are not the same as those imposed by the real data, as the real data have narrower operating margins. We concluded that the alternation of class 0's and class 1's is extremely important in the performance of online training. The existence of 0's and 1's distribution threshold forced the search and characterisation of the randomness of the data that were used in online training. This study showed that the distribution of classes in time is an obstacle to online training, thus hindering the conventional online training.

Chapter 7

Conclusions

The work introduced and exposed on the previous chapters of this dissertation has been developed with the objective of creating an intrusion detection system and determining a) if command and control traffic evolves in time; and b) if the detection model can adapt to the changes of command and control traffic over time.

Chapter 3 shows that the architecture adopted for the detection system achieved all the operational objectives.

In Chapter 5, we concluded that traffic characteristics change over time. However, we found that the detection model was able to adapt to the changing characteristics over time, leading to the conclusion that incremental offline training allows an adaptation to the changes in traffic characteristics.

The results obtained in Chapter 6 show that is possible to use online training to update the detection model. Nevertheless, after performing some tests we concluded that online training has limitations. Online training performance depends on the natural classification distribution of the traffic (0's and 1's distribution), which it could lead to failure the online training. In summary, we conclude that it is necessary to create countermeasures to allow online training, which we will describe in the future work section.

This thesis and its results confirm that command and control traffic evolves in time and the detection model can adapt to the changes of command and control traffic over time.

7.1 Future Work

The future work will pass to replace offline training by online training on the intrusion detection system. However, it is necessary to find a method that allows online training and mitigates the 0's and 1's distribution problem.

Another direction is to evolve the IDS to an IPS, this implies that the system will start using only the characteristics of the beginning of the flow instead of the complete flow characteristics (since an IPS immediately blocks the flows considered malignant). This change will have a major implication for the detection system, since until now the system used the complete flow features

to train the detection model. However, IPS prevents the training with the complete flow features since it immediately blocks the flows considered malignant. Therefore, it is necessary to train the detection models only with the initial characteristics of the flow. This way, it will be necessary to prove that detection is feasible using only the initial features of the flow.

Appendix A

Intrusion Detection System

A.1 Full Description Tstat Feature Set

The full description of Tstat's features follows in the following tables.

C2S	S2C	Short description	Unit	Long description
1	15	Client/Server IP addr	-	IP addresses of the client/server
2	16	Client/Server TCP port	-	TCP port addresses for the client/server
3	17	packets	-	total number of packets observed form the client/server
4	18	RST sent	0/1	0 = no RST segment has been sent by the client/server
5	19	ACK sent	-	number of segments with the ACK field set to 1
6	20	PURE ACK sent	-	number of segments with ACK field set to 1 and no data
7	21	unique bytes	bytes	number of bytes sent in the payload
8	22	data pkts	-	number of segments with payload
9	23	data bytes	bytes	number of bytes transmitted in the payload, including retransmissions
10	24	rexmit pkts	-	number of retransmitted segments
11	25	rexmit bytes	bytes	number of retransmitted bytes
12	26	out seq pkts	-	number of segments observed out of sequence
13	27	SYN count	-	number of SYN segments observed (including rtx)
14	28	FIN count	-	number of FIN segments observed (including rtx)
29		First time abs	ms	Flow first packet absolute time (epoch)
30		Last time abs	ms	Flow last segment absolute time (epoch)
31		Completion time	ms	Flow duration since first packet to last packet
32		C first payload	ms	Client first segment with payload since the first flow segment
33		S first payload	ms	Server first segment with payload since the first flow segment
34		C last payload	ms	Client last segment with payload since the first flow segment
35		S last payload	ms	Server last segment with payload since the first flow segment
36		C first ack	ms	Client first ACK segment (without SYN) since the first flow segment
37		S first ack	ms	Server first ACK segment (without SYN) since the first flow segment
38		C Internal	0/1	1 = client has internal IP, 0 = client has external IP
39		S Internal	0/1	1 = server has internal IP, 0 = server has external IP
40		C anonymized	0/1	1 = client IP is CryptoPAn anonymized
41		S anonymized	0/1	1 = server IP is CryptoPAn anonymized
42		Connection type	-	Bitmap stating the connection type as identified by TCPL7 inspection engine (see protocol.h)
43		P2P type	-	Type of P2P protocol, as identified by the IPP2P engine (see ipp2p_tstat.h)
44		HTTP type	-	For HTTP flows, the identified Web2.0 content (see the http_content enum in struct.h)

Table A.1: Core/Basic TCP Set

C2S	S2C	Short description	Unit	Long description
45	52	Average rtt	ms	Average RTT computed measuring the time elapsed between the data segment and the corresponding ACK
46	53	rtt min	ms	Minimum RTT observed during connection lifetime
47	54	rtt max	ms	Maximum RTT observed during connection lifetime
48	55	Stdev rtt	ms	Standard deviation of the RTT
49	56	rtt count	-	Number of valid RTT observation
50	57	ttl_min	-	Minimum Time To Live
51	58	ttl_max	-	Maximum Time To Live

Table A.2: TCP End to End Set

C2S	S2C	Short description	Unit	Long description
X	X+23	RFC1323 ws	0/1	Window scale option sent
X+1	X+24	RFC1323 ts	0/1	Timestamp option sent
X+2	X+25	window scale	-	Scaling values negotiated [scale factor]
X+3	X+26	SACK req	0/1	SACK option set
X+4	X+27	SACK sent	-	number of SACK messages sent
X+5	X+28	MSS	bytes	MSS declared
X+6	X+29	max seg size	bytes	Maximum segment size observed
X+7	X+30	min seg size	bytes	Minimum segment size observed
X+8	X+31	win max	bytes	Maximum receiver window announced (already scale by the window scale factor)
X+9	X+32	win min	bytes	Maximum receiver windows announced (already scale by the window scale factor)
X+10	X+33	win zero	-	Total number of segments declaring zero as receiver window
X+11	X+34	cwin max	bytes	Maximum in-flight-size computed as the difference between the largest sequence number so far, and the corresponding last ACK message on the reverse path. It is an estimate of the congestion window
X+12	X+35	cwin min	bytes	Minimum in-flight-size
X+13	X+36	initial cwin	bytes	First in-flight size, or total number of unack-ed bytes sent before receiving the first ACK segment
X+14		rtx RTO	-	Number of retransmitted segments due to timeout expiration
X+15		rtx FR	-	Number of retransmitted segments due to Fast Retransmit (three dup-ack)
X+16		reordering	-	Number of packet reordering observed
X+17		net dup	-	Number of network duplicates observed
X+18		unknown	-	Number of segments not in sequence or duplicate which are not classified as specific events
X+19		flow control	-	Number of retransmitted segments to probe the receiver window
X+20		unnece rtx RTO	-	Number of unnecessary transmissions following a timeout expiration
X+21		unnece rtx FR	-	Number of unnecessary transmissions following a fast retransmit
X+22		!= SYN seqno	0/1	1 = retransmitted SYN segments have different initial seqno

Table A.3: TCP Options Set

C2S	S2C	Short description	Unit	Long description
K		HTTP Request count	-	Number of HTTP Requests (GET/POST/HEAD) seen in the C2S direction (for HTTP connections)
K+1		HTTP Response count	-	Number of HTTP Responses (HTTP) seen in the S2C direction (for HTTP connections)
K+2		First HTTP Response	-	First HTTP Response code seen in the server->client communication (for HTTP connections)
K+3		PSH-separated C2S	-	number of push separated messages C2S
K+4		PSH-separated S2C	-	number of push separated messages S2C
K+5		TLS Client Hello SNI	-	For TLS flows, the server name indicated by the client in the Hello message extensions
K+6		TLS Server Hello SCN	-	For TLS flows, the subject CN name indicated by the server in its certificate
K+7		TLS Client NPN/ALPN	-	For TLS flows, a bitmap representing the usage of NPN/ALPN for HTTP2/SPDY negotiation
K+8		TLS Server NPN/ALPN	-	For TLS flows, a bitmap representing the usage of NPN/ALPN for HTTP2/SPDY negotiation
K+9		TLS Client ID reuse	-	For TLS flows, indicates that the Client Hello carries an old Session ID
K+10		TLS Client Last Handshake	ms	For TLS flows, time of Client last packet seen before first Application Data (relative)
K+11		TLS Server Last Handshake	ms	For TLS flows, time of Server last packet seen before first Application Data (relative)
K+12		TLS Client App Data Time	ms	For TLS flows, time between the Client first Application Data message and the first flow segment
K+13		TLS Server App Data Time	ms	For TLS flows, time between the Server first Application Data message and the first flow segment
K+14		TLS Client App Data Bytes	bytes	For TLS flows, relative sequence number for the Client first Application Data message
K+15		TLS Server App Data Bytes	bytes	For TLS flows, relative sequence number for the Client first Application Data message
K+16		FQDN	-	Fully Qualified Domain Name recovered using DNHunter
K+17		IP of DNS resolver	-	IP address of the contacted DNS resolver
K+18		DNS request time	ms	unixtime (in ms) of the DNS request
K+19		DNS response time	ms	unixtime (in ms) of the DNS response

Table A.4: TCP Layer 7 Set

A.2 Code

This section is for the demonstration of some scripts used in the thesis.

```
#!/bin/bash
#   Run on sudo su  ROOT needed

helpFunction()
{
 echo ""
 echo "Usage: ./tap name  ex tap0"
 echo ""
 exit 1 # Exit script after printing help
}

# Print helpFunction in case parameters are empty
if [ -z "$1" ]
then
 echo "";
 echo "Some or all of the parameters are empty";
 helpFunction
fi

sudo tunctl -t $1
sudo ifconfig $1 172.16.0.1 netmask 255.255.255.0 up
sudo ifconfig $1 up
```

Listing A.1: Creation of TAP interface

```
#!/bin/bash
# -K option will preload the PCAP file into memory before testing
# -t option will send as quickly as possible
# --loop to add a loop not implemented

helpFunction()
{
 echo ""
 echo "Usage: pcapreplay path_to_pcap  eth_interface velocity(Mbps)"
 echo ""
 exit 1 # Exit script after printing help
}

# Print helpFunction in case parameters are empty
if [ -z "$1" ] || [ -z "$2" ] || [ -z "$3" ]
then
 echo "";
 echo "Some or all of the parameters are empty";
 helpFunction
fi

#tcpreplay -i tap0 -M 10 ../pcap/101.pcap
#sudo tcpreplay -i $2 -M $3 -K $1
sudo tcpreplay -i $2 -K $1
```

Listing A.2: Pcap Replay

A.3 Testing

The Figures in this section A.3 refer to the test performed in subsection 3.5.

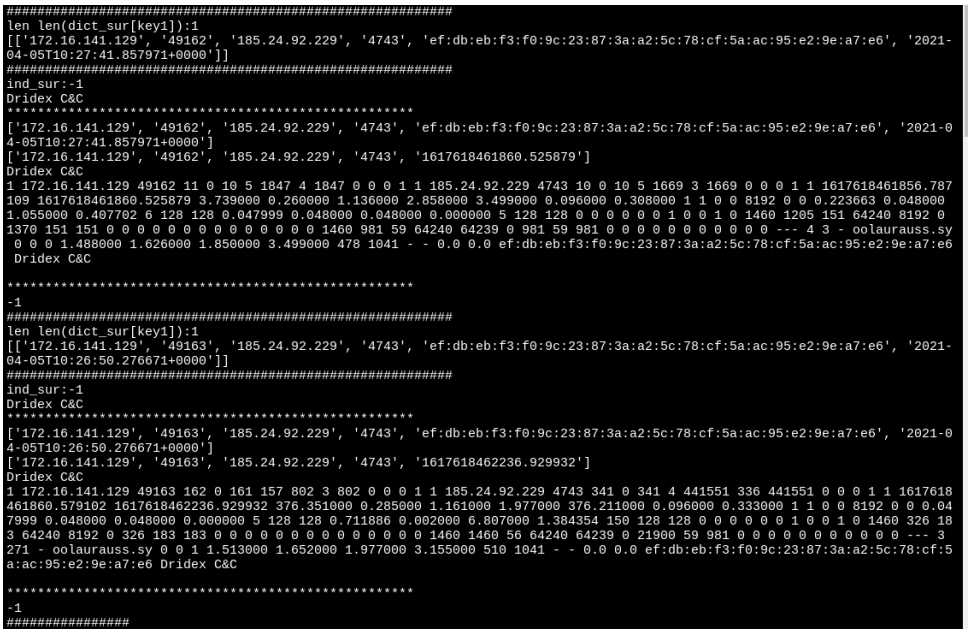


Figure A.1: Alert Malware

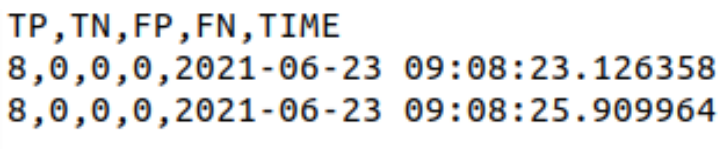
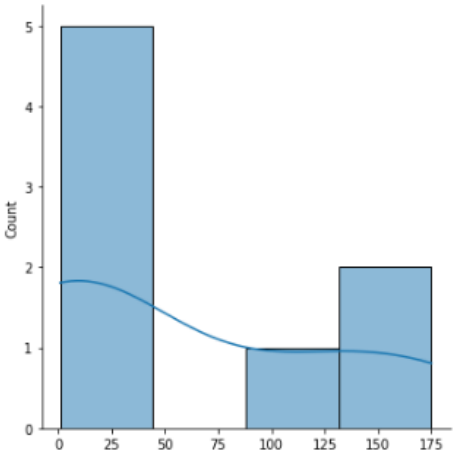


Figure A.2: Log Matrix

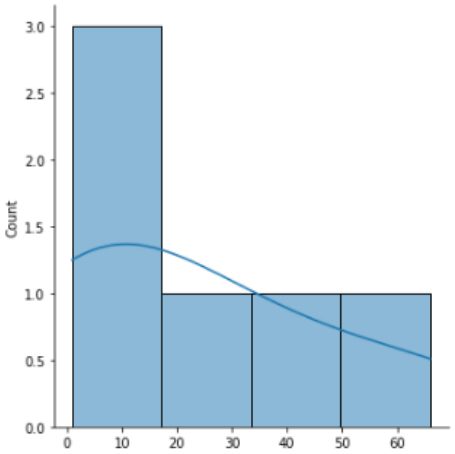
Figure A.3: Program's Log

A.4 Distribution Studies 0's and 1's per Year

The Figures in this section A.4 refer to the studies performed in subsection 6.2.3. Each set of figures represents one year and contains the distribution of 0's and 1's.

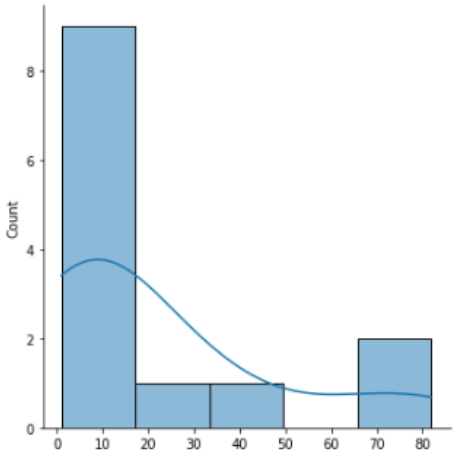


(a) 0's year 2014

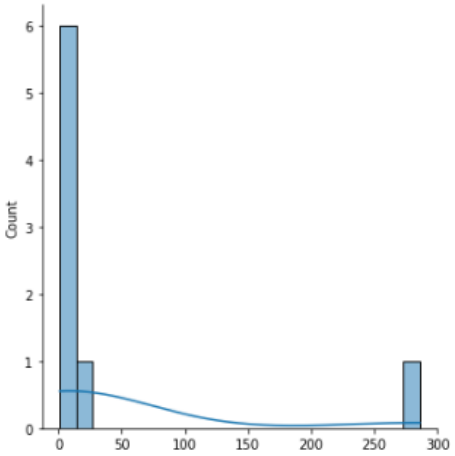


(b) 1's year 2014

Figure A.4: Year 2014

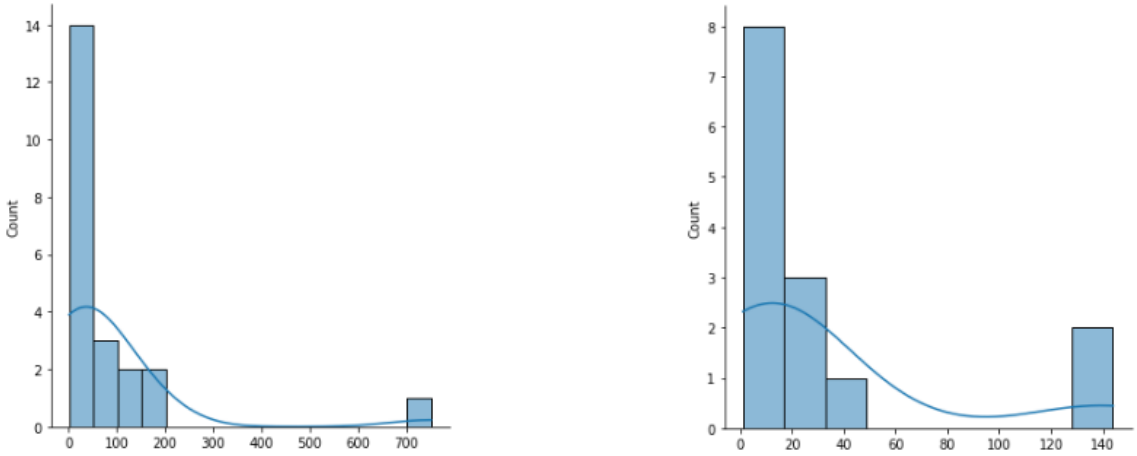


(a) 0's year 2016



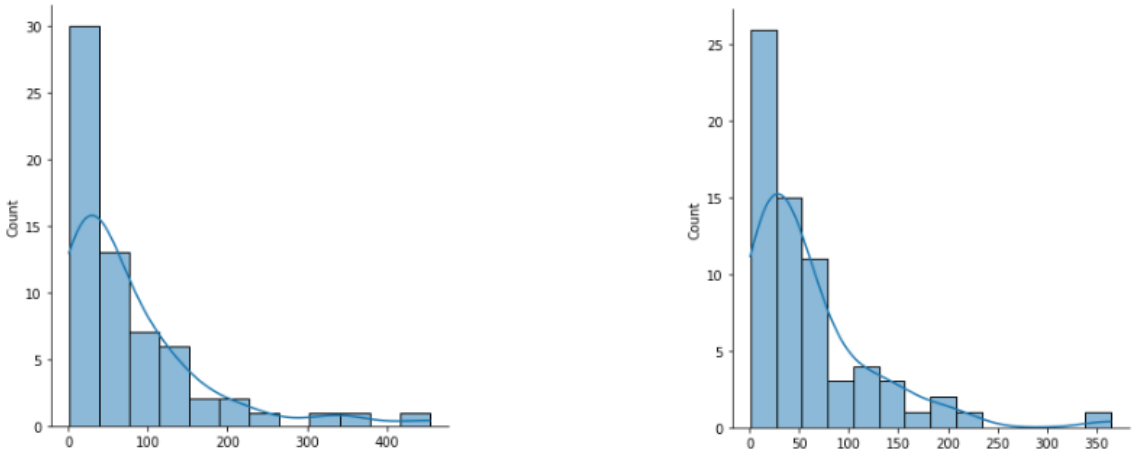
(b) 1's year 2016

Figure A.5: Year 2016



(a) 0's year 2017 (b) 1's year 2017

Figure A.6: Year 2017



(a) 0's year 2018 (b) 1's year 2018

Figure A.7: Year 2018

A.5 Limitation of 0's and 1's

The Figures in this section A.5 refer to the studies carried out in subsection 6.2.2. Each set of Figures, contains 3 metrics (precision, recall, f1).

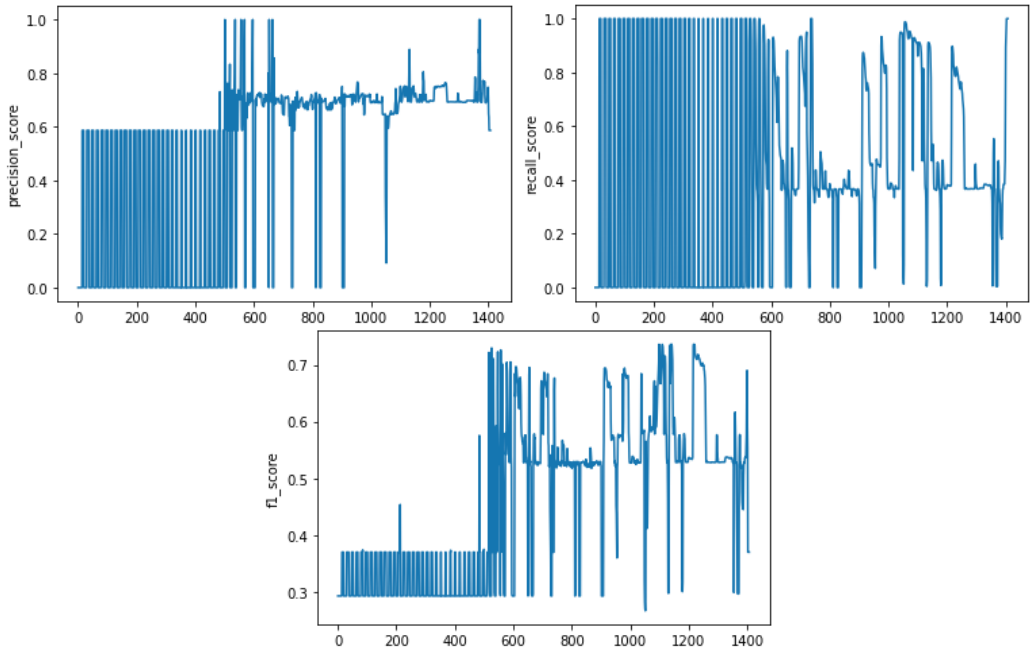


Figure A.8: Study One - 128/32 (0's/1's)

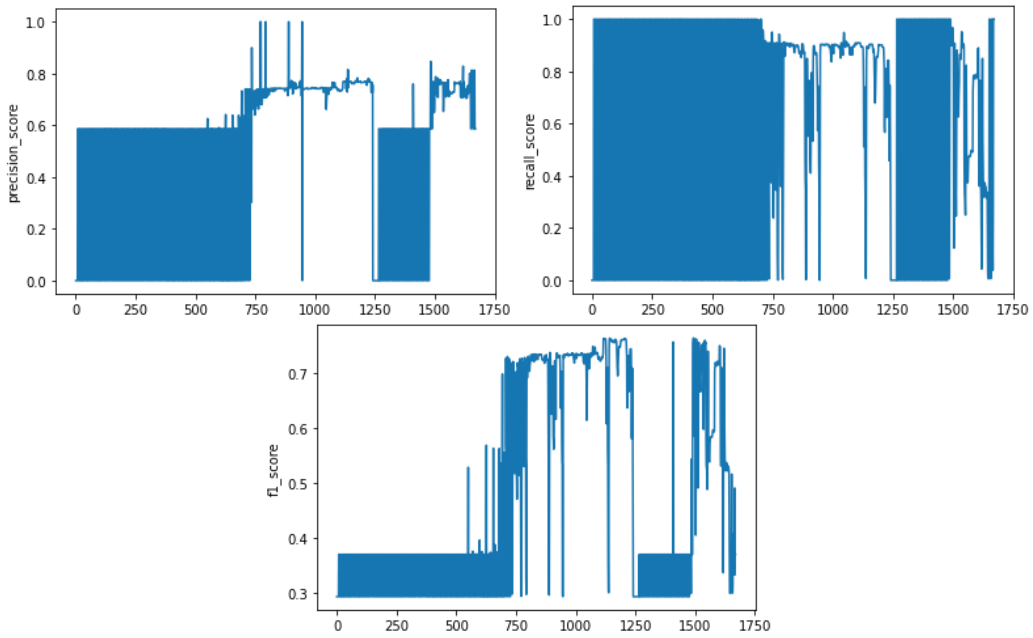


Figure A.9: Study Two - 64/32 (0's/1's)

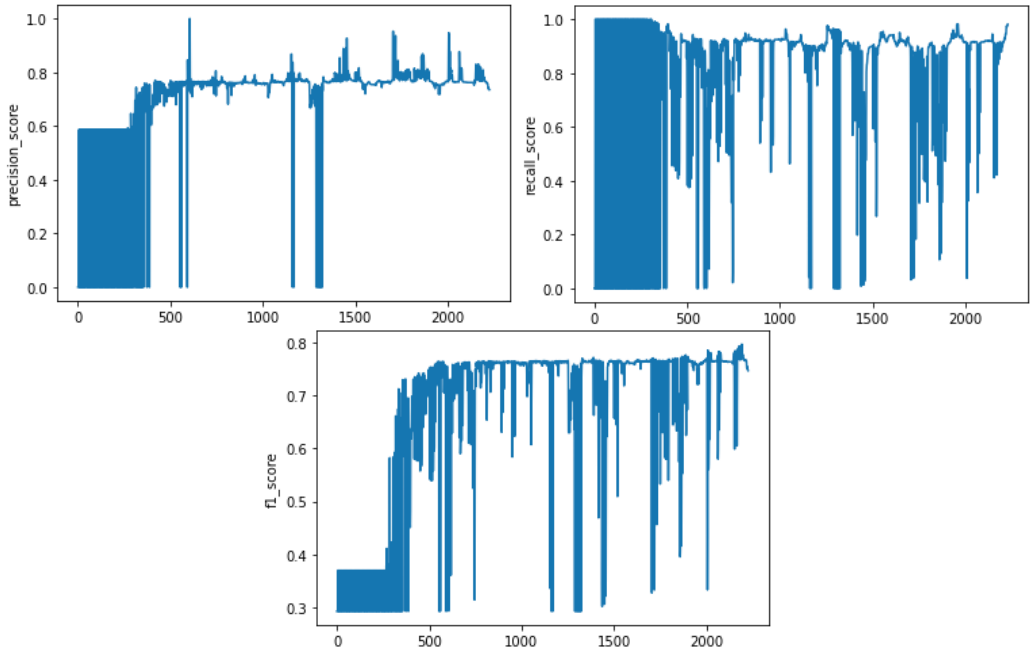


Figure A.10: Study Three - 32/32 (0's/1's)

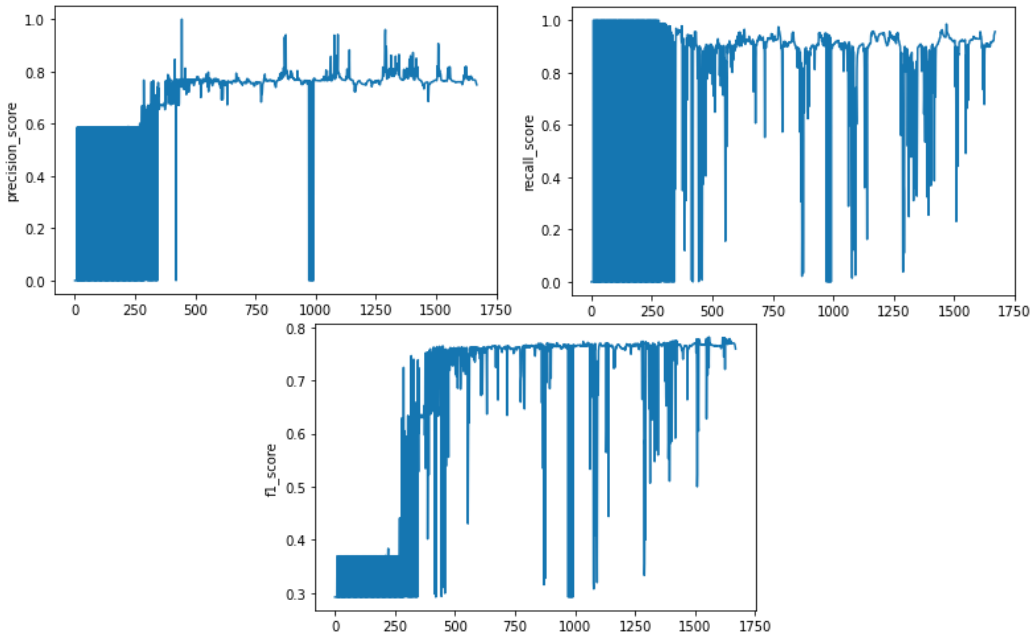


Figure A.11: Study Four - 32/16 (0's/1's)

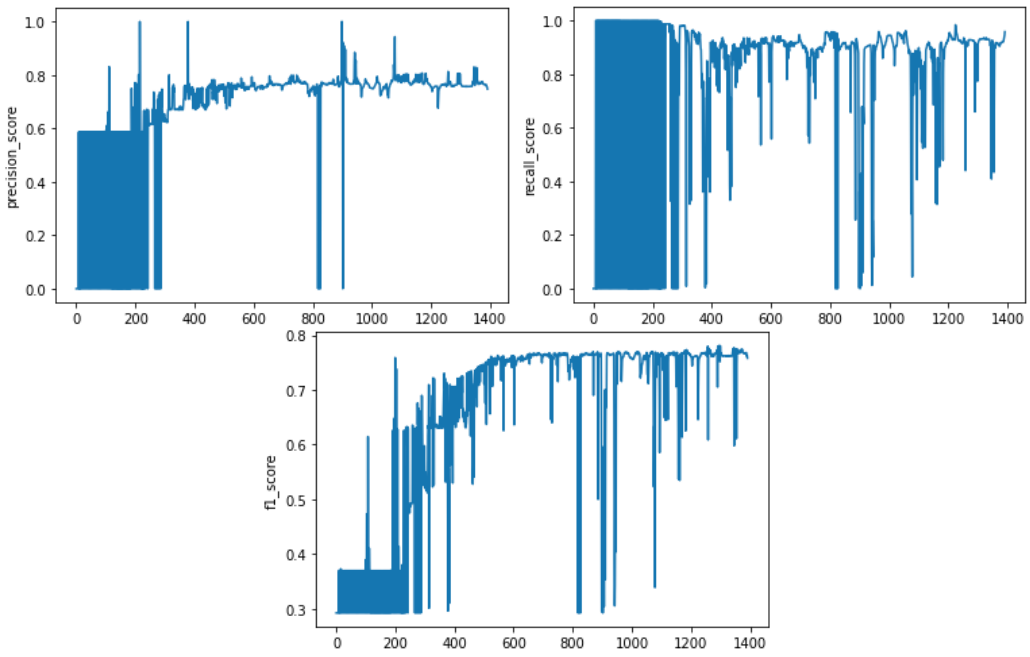


Figure A.12: Study Five - 32/8 (0's/1's)

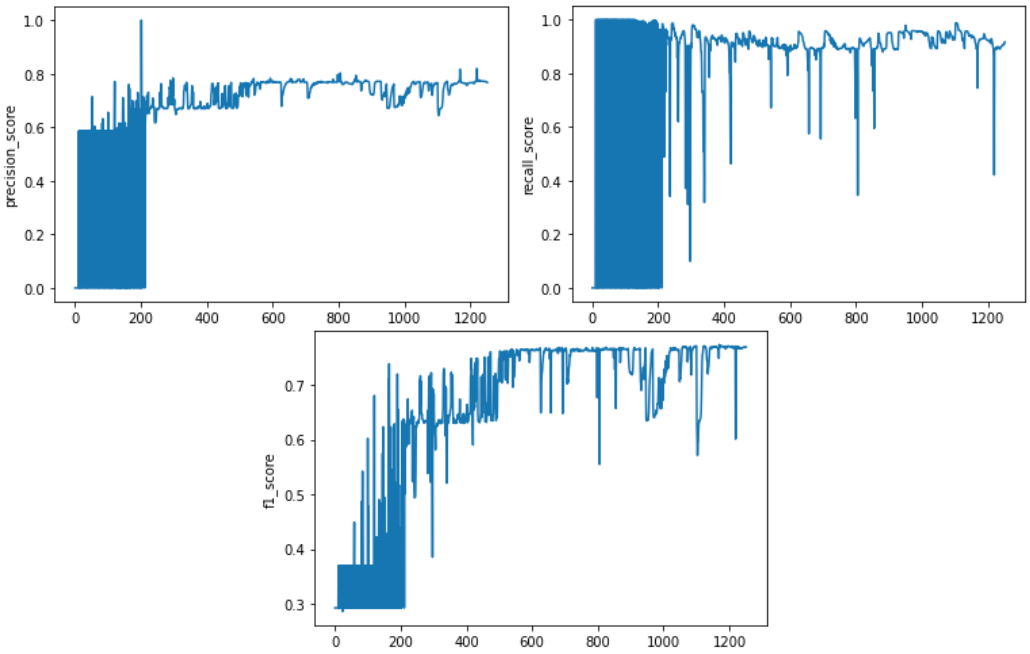


Figure A.13: Study Six - 32/4 (0's/1's)

References

[1] F. Hock and P. Kortiš. Commercial and open-source based intrusion detection system and intrusion prevention system (IDS/IPS) design for an IP networks. In *2015 13th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 1–4, 2015. doi:10.1109/ICETA.2015.7558466.

[2] R. Zheng, J. Liu, K. Li, S. Liao, and L. Liu. Detecting malicious TLS network traffic based on communication channel features. In *2020 IEEE 8th International Conference on Information, Communication and Networks (ICICN)*, pages 14–19, 2020. doi:10.1109/ICICN51133.2020.9205087.

[3] B. Anderson and D. McGrew. Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 1723–1732, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3097983.3098163.

[4] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng. Malware traffic classification using convolutional neural network for representation learning. In *2017 International Conference on Information Networking (ICOIN)*, pages 712–717, 2017. doi:10.1109/ICOIN.2017.7899588.

[5] M. Trevisan, A. Finamore, M. Mellia, M. Munafo, and D. Rossi. Traffic analysis with off-the-shelf hardware: Challenges and lessons learned. *IEEE Communications Magazine*, 55(3):163–169, 2017. doi:10.1109/MCOM.2017.1600756CM.

[6] V. Losing, B. Hammer, and H. Wersing. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, 275:1261 – 1274, 2018. doi:https://doi.org/10.1016/j.neucom.2017.06.084.

[7] I. Lee, H. Roh, and W. Lee. Poster abstract: Encrypted malware traffic detection using incremental learning. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1348–1349, 2020. doi:10.1109/INFOCOMWKSHPS50562.2020.9162971.

[8] H. Anderson. Evading machine learning malware detection. 2017.

[9] L. Chen, Y. Ye, and T. Bourlai. Adversarial machine learning in malware detection: Arms race between evasion attack and defense. In *2017 European Intelligence and Security Informatics Conference (EISIC)*, pages 99–106, 2017. doi:10.1109/EISIC.2017.21.

[10] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel. Disclosure: Detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the*

28th Annual Computer Security Applications Conference, ACSAC '12, page 129–138, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2420950.2420969.

[11] A. Borkar, A. Donode, and A. Kumari. A survey on intrusion detection system (IDS) and internal intrusion detection and protection system (IIDPS). In *2017 International Conference on Inventive Computing and Informatics (ICICI)*, pages 949–953, 2017. doi:10.1109/ICICI.2017.8365277.

[12] S. A. R. Shah and B. Issac. Performance comparison of intrusion detection systems and application of machine learning to snort system. *Future Generation Computer Systems*, 80:157 – 170, 2018. doi:https://doi.org/10.1016/j.future.2017.10.016.

[13] D. A. Bhosale and V. M. Mane. Comparative study and analysis of network intrusion detection tools. In *2015 International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, pages 312–315, 2015. doi:10.1109/ICATCCT.2015.7456901.

[14] H. Hindy, D. Brosset, E. Bayne, A. K. Seeam, C. Tachtatzis, R. Atkinson, and X. Bellekens. A taxonomy of network threats and the effect of current datasets on intrusion detection systems. *IEEE Access*, 8:104650–104675, 2020. doi:10.1109/ACCESS.2020.3000179.

[15] V. Méndez-García, P. Jiménez-Ramírez, M. Á. Meléndez-Ramírez, F. M. Torres-Martínez, R. Llamas-Contreras, and H. González. Comparative analysis of banking malware. In *2014 IEEE Central America and Panama Convention (CONCAPAN XXXIV)*, pages 1–5, 2014. doi:10.1109/CONCAPAN.2014.7000412.

[16] R. Branco and U. Shamir. Architecture for automation of malware analysis. In *2010 5th International Conference on Malicious and Unwanted Software*, pages 106–112, 2010. doi:10.1109/MALWARE.2010.5665786.

[17] P. Black and J. Opacki. Anti-analysis trends in banking malware. In *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 1–7, 2016. doi:10.1109/MALWARE.2016.7888738.

[18] Z. Berkay Celik, R. J. Walls, P. McDaniel, and A. Swami. Malware traffic detection using tamper resistant features. In *MILCOM 2015 - 2015 IEEE Military Communications Conference*, pages 330–335, 2015. doi:10.1109/MILCOM.2015.7357464.

[19] S. Silva, R. Silva, R. Pinto, and R. Salles. Botnets: A survey. *Computer Networks*, 57(2):378 – 403, 2013. Botnet Activity: Analysis, Detection and Shutdown. doi:https://doi.org/10.1016/j.comnet.2012.07.021.

[20] D. Zhao, I. Traore, B. Sayed, W. Lu, S. Saad, A. Ghorbani, and D. Garant. Botnet detection based on traffic behavior analysis and flow intervals. *Computers & Security*, 39:2 – 16, 2013. 27th IFIP International Information Security Conference. doi:https://doi.org/10.1016/j.cose.2013.04.007.

[21] F. V. Alejandre, N. C. Cortés, and E. A. Anaya. Feature selection to detect botnets using machine learning algorithms. In *2017 International Conference on Electronics, Communications and Computers (CONIELECOMP)*, pages 1–7, 2017. doi:10.1109/CONIELECOMP.2017.7891834.

[22] K. C. Pai, S. Mitra, and M. Chari S. Novel TLS signature extraction for malware detection. In *2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pages 1–3, 2020. doi:10.1109/CONECCT50063.2020.9198590.

[23] O. Roques. Detecting malware in TLS traffic, PhD Thesis, Imperial College London. 2019.

[24] D. Han, Z. Wang, Y. Zhong, W.i Chen, J.i Yang, S. Lu, X. Shi, and X. Yin. Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors, 2020. arXiv:2005.07519.

[25] D. Kim, J. Han, J. Lee, H. Roh, and W. Lee. Poster: Feasibility of malware traffic analysis through TLS-encrypted flow visualization. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–2, 2020. doi:10.1109/ICNP49622.2020.9259387.

[26] R. Mellia, M. Cigno and F. Neri. Measuring IP and TCP behavior on edge nodes with tstat. *Computer Networks*, 47:1–21, 01 2005. doi:10.1016/j.comnet.2004.06.026.

[27] F. Iglesias and T. Zseby. Analysis of network traffic features for anomaly detection. *Machine Learning*, 101(1):59–84, Oct 2015. doi:10.1007/s10994-014-5473-9.

[28] S. Ni, Q. Qian, and R. Zhang. Malware identification using visualization images and deep learning. *Computers & Security*, 77:871 – 885, 2018. doi:https://doi.org/10.1016/j.cose.2018.04.005.

[29] H. Naeem, B. Guo, M. Naeem, F. Ullah, H. Aldabbas, and M. Javed. Identification of malicious code variants based on image visualization. *Computers & Electrical Engineering*, 76:225 – 237, 2019. doi:https://doi.org/10.1016/j.compeleceng.2019.03.015.

[30] S. Zanero. Flaws and frauds in the evaluation of IDS/IPS technologies. In *Proc. of FIRST*. Citeseer, 2007.

[31] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. 01 2018. doi:10.14722/ndss.2018.23211.

[32] Politecnico di Torino. Tstat: tstat.polito.it, 2021. URL: http://https://http://tstat.polito.it/.

[33] Open Information Security Foundation. Suricata: suricata.io, 2021. URL: https://suricata.io/.

[34] C. Novo. Adversarial malware command and control traffic generation, Master's thesis, Faculdade de Engenharia, Universidade do Porto. 2020.

[35] Malware Traffic Analysis. MTA: malware-traffic-analysis.net, 2021. URL: http://https://www.malware-traffic-analysis.net/.

[36] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. arXiv:1412.6980.

[37] Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels, 2018. arXiv:1805.07836.