Adaptive Secure Pipeline for Attacks Detection in Networks with set of Distribution Hosts

خط أنابيب آمن متكيف لاكتشاف الهجمات في الشبكات مع مجموعة من مضيفي التوزيع

by

SUROUR ALSHAMSI

Dissertation submitted in fulfilment

of the requirements for the degree of

MSc INFORMATICS

at

The British University in Dubai

January 2022

DECLARATION

I warrant that the content of this research is the direct result of my own work and that any use made in it of published or unpublished copyright material falls within the limits permitted by international copyright conventions.

I understand that a copy of my research will be deposited in the University Library for permanent retention.

I hereby agree that the material mentioned above for which I am author and copyright holder may be copied and distributed by The British University in Dubai for the purposes of research, private study or education and that The British University in Dubai may recover from purchasers the costs incurred in such copying and distribution, where appropriate.

I understand that The British University in Dubai may make a digital copy available in the institutional repository.

I understand that I may apply to the University to retain the right to withhold or to restrict access to my thesis for a period which shall not normally exceed four calendar years from the congregation at which the degree is conferred, the length of the period to be specified in the application, together with the precise reasons for making that application.

Signature of the student

COPYRIGHT AND INFORMATION TO USERS

Abstract

Currently, malware continues to represent one of the main computer security threats. It is difficult to have efficient detection systems to precisely separate normal behavior from malicious behavior, based on the analysis of network traffic. This is due to the characteristics of malicious and normal traffic, since normal traffic is very complex, diverse and changing; and malware is also changeable, migrates and hides itself pretending to be normal traffic.

In addition, there is a large amount of data to analyze and the detection is required in real time to be useful. It is therefore necessary to have an effective mechanism to detect malware and attacks on the network. In order to benefit from multiple different classifiers, and exploit their strengths, the use of ensembling algorithms arises, which combine the results of the individual classifiers into a final result to achieve greater precision and thus a better result. This can also be applied to cybersecurity problems, in particular to the detection of malware and attacks through the analysis of network traffic, a challenge that we have raised in this thesis.

The research work carried out, in relation to attack detection ensemble learning, mainly aims to increase the performance of machine learning algorithms by combining their results. Most of the studies propose the use of some technique, existing ensemble learning or created by the authors, to detect some type of attack in particular and not attacks in general. So far none addresses the use of Threat Intelligence (IT) data in Ensemble Learning algorithms to improve the detection process, nor does it work as a function of time, that is, taking into account what happens on the network in a limited time interval. The objective of this thesis is to propose a methodology to apply ensembling in the detection of infected hosts considering these two aspects.

As a function of the proposed objective, ensembling algorithms applicable to network security have been investigated and evaluated, and a methodology for detecting infected hosts using ensembling has been developed, based on experiments designed and tested

with real datasets. This methodology proposes to carry out the process of detecting infected hosts in three phases. These phases are carried out each a certain amount of time.

Each of them applies ensembling with different objectives. The first phase is done to classify each network flow belonging to the time window, as malware or normal. The second phase applies it to classify the traffic between an origin and a destination, as malicious or normal, indicating whether it is part of an infection. And finally, the third phase, in order to classify each host as infected or not infected, considering the hosts that originate the communications.

The implementation in phases allows us to solve, in each one of them, one aspect of the problem, and in turn take the predictions of the previous phase, which are combined with the analysis of the phase itself to achieve better results. In addition, it implies carrying out the training and testing process in each phase. Since the best model is obtained from training, each time it is performed for a given phase, the model is adjusted to detect new attacks. This represents an advantage over tools based on firm rules or static rules, where you have to know the behavior to add new rules.

الملخص

في الوقت الحالي، لا تزال البرامج الضارة تمثل أحد التهديدات الرئيسية لأمان الكمبيوتر. من الصعب أن يكون لديك أنظمة كشف فعالة لفصل السلوك الطبيعي بدقة عن السلوك الضار، بناءً على تحليل حركة مرور الشبكة. ويرجع ذلك إلى خصائص حركة المرور الخبيثة والعادية، لأن حركة المرور العادية معقدة للغاية ومتنوعة ومتغيرة؛ والبرامج الضارة قابلة للتغيير أيضًا، وتهاجر وتخفي نفسها متظاهرة بأنها حركة مرور عادية.

بالإضافة إلى ذلك، هناك قدر كبير من البيانات لتحليلها والكشف مطلوب في الوقت الفعلي ليكون مفيدًا. لذلك من الضروري وجود آلية فعالة لاكتشاف البرامج الضارة والهجمات على الشبكة. من أجل الاستفادة من العديد من المصنفات المختلفة، واستغلال نقاط قوتها، ينشأ استخدام خوارزميات التجميع، والتي تجمع نتائج المصنفات الفردية في نتيجة نهائية لتحقيق دقة أكبر وبالتالي نتيجة أفضل. يمكن أيضًا تطبيق هذا على مشاكل الأمن السيبراني، لا سيما للكشف عن البرامج الضارة والهجمات من خلال تحليل حركة مرور الشبكة، وهو التحدي الذي طرحناه في هذه الأطروحة.

يهدف العمل البحثي الذي تم إجراؤه، فيما يتعلق بتعلم مجموعة اكتشاف الهجمات، بشكل أساسي إلى زيادة أداء خوارزميات التعلم الآلي من خلال الجمع بين نتائجها. تقترح معظم الدراسات استخدام بعض الأساليب، أو التعلم الجماعي الحالي أو الذي ابتكره المؤلفون، لاكتشاف نوع من الهجوم على وجه الخصوص وليس الهجمات بشكل عام. Ensemble في خوارزميات (IT) حتى الآن، لا يوجد شيء يتناول استخدام بيانات استخبارات التهديدات لتحسين عملية الكشف، كما أنه لا يعمل كوظيفة زمنية، أي مع الأخذ في الاعتبار ما يحدث على الشبكة Learning في فترة زمنية محدودة. الهدف من هذه الرسالة هو اقتراح منهجية لتطبيق التجميع في الكشف عن العوائل المصابة مع مراعاة هذين الجانبين.

كدالة للهدف المقترح، تم فحص وتقييم خوارزميات التجميع المطبقة على أمان الشبكة، وتم تطوير منهجية للكشف عن المضيفين المصابين باستخدام التجميع، بناءً على التجارب المصممة والمختبرة بمجموعات بيانات حقيقية. تقترح هذه المنهجية إجراء عملية الكشف عن العوائل المصابة على ثلاث مراحل. يتم تنفيذ هذه المراحل في كل فترة زمنية معينة.

كل واحد منهم يطبق التجميع بأهداف مختلفة. تتم المرحلة الأولى لتصنيف كل تدفق شبكة ينتمي إلى النافذة الزمنية، على أنه برامج ضارة أو عادية. يتم تطبيقه في المرحلة الثانية لتصنيف حركة المرور بين الأصل والوجهة، على أنها ضارة أو عادية، مع الإشارة إلى ما إذا كانت جزءًا من إصابة. وأخيرًا، المرحلة الثالثة، من أجل تصنيف كل مضيف على أنه مصاب أو غير مصاب، مع الأخذ في الاعتبار المضيفين الذين أنشأوا الاتصالات.

يتيح لنا التنفيذ على مراحل أن نحل، في كل منها، جانبًا واحدًا من المشكلة، وبالتالي نأخذ تنبؤات المرحلة السابقة، والتي يتم دمجها مع تحليل المرحلة نفسها لتحقيق نتائج أفضل. بالإضافة إلى ذلك، فإنه يعني تنفيذ عملية التدريب والاختبار في كل مرحلة. نظرًا لأنه يتم الحصول على أفضل نموذج من التدريب، في كل مرة يتم تنفيذه لمرحلة معينة، يتم تعديل النموذج لاكتشاف هجمات جديدة. يمثل هذا ميزة على الأدوات القائمة على قواعد ثابتة أو قواعد ثابتة، حيث يتعين عليك معرفة السلوك لإضافة قواعد جديدة.

Acknowledgements

Thankful to God Almighty for everything.

Thanks to Dr. Khaled Shaalan, Dr. Sherief Abdallah, and Dr. Cornelius Ncube. Who was very generous in reaching the best results and achieving the desired goal.

Thanks to my family, my father, mother, brothers, and sisters, who were with me and supported me at all moments to accomplish what started.

This research report is dedicated to all of them, and everyone wants to use it for their benefit.

Table of Contents

Table of Figures

Table of Tables

Abbreviation

TI	Threat Intelligence
LR	Logistic Regression
RF	Random Forest
NB	Naive Bayes
KNN	K-Nearest Neighbor
DT	Decision Tree
IoT	Internet of Things
IDS	Intrusion Detection Systems
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

Chapter One: Introduction

1. 1. Introduction and Background

In the cybersecurity field, professionals and researchers have designed a variety of cyber defense systems over the years in order to protect organizations from malicious attackers. These systems face threats such as viruses, Trojans, worms and botnets among others. Existing solutions based on Intrusion Detection Systems (IDS) include proactive approaches to anticipate vulnerabilities in computer systems and thus be able to carry out mitigation actions. However, over the years the number of threats has increased enormously, especially due to the appearance of malware development environments capable of almost automatically generating different versions of the same virus, making it possible for any hobbyist to produce your own variation. This proliferation of malware makes the rule databases used by IDS larger and larger, thus increasing the computation time required for such detection.

In particular, these detection systems currently have three weak points that are worth highlighting:

- The necessary balance between the effectiveness of threat detection and the speed at which the data can be examined collected in modern data networks;
- The need to be able to detect such threats even when the data travels encrypted.
- The difficulty of detecting new versions of malware even if they are variations of an already known family.

The first is motivated by the transfer speeds that communication networks are reaching (above 10 Gbps), with increasing data exchange volumes. To this must be added the new fifth generation (5G) mobile technology, which promises to provide latency and transfer speed never seen before in wireless networks, allowing unprecedented expansion of the Internet of Things (IoT). All this will make it really difficult to capture and analyze every packet that circulates on the network, causing the current detection procedures to become

obsolete if we are not able to adapt them properly. Precisely because of the ubiquity of Internet access, partly thanks to wireless technologies, and the enormous growth of the IoT, there are millions of devices with vulnerabilities that can be used to form botnets.

A botnet is a set of devices connected to the Internet that an attacker previously infected with software that allows him to manage them remotely to carry out all kinds of actions, such as distributed denial of service attacks, information theft. Sensitive or critical such as bank accounts and personal data, or even theft of CPU cycles to mine cryptocurrencies. Botnets have become one of the great current and future security problems in data networks; Suffice it to mention that, according to the recent Nokia Threat Intelligence Report, in 2018 IoT-based botnets accounted for 78% of detected malware attacks and 16% of infected IoT devices [1].

Therefore, in the context of 5G and IoT communications, the detection of malware becomes a challenge due to the increasing diversity of said malware, the rules of which must be applied to each packet, since the transfer rates so high along with the great volume of data that moves, they leave little time to examine each packet that circulates on the network. When we evaluate the volume of packets that current deep packet inspection tools can handle, we find that the well-known Snort supports wired networks of up to 1 Gbps, starting to discard packets due to overload from 1.5 Gbps [2]. This has led to the appearance of hardware solutions based on programmable gate arrays [3] or specific purpose integrated circuits, which allow working with speeds of up to 7.2 Gbps [4] . Even so, these speeds are far from those expected in the near future. Due in part to this, IDS-based detection solutions have had to evolve from analyzing network packets to analyzing network traffic flows using new techniques based on artificial intelligence [5] . For example, a block-based neural network model used in a stream anomaly-based IDS was able to handle 22 Gbps traffic using programmable gate arrays FPGAs [6]. A complete review of solutions to quickly classify network flows and detect attacks or malicious code can be found in [7].

With regard to the second weak point, an increasing number of malware encrypts its communications [8], making deep examination of packets impossible and common detection tools ineffective. To this must be added the progressive increase in the amount

of encrypted traffic in everyday communications, and the obligation to use such encryption in environments where privacy is critical, such as for example medical environments. These environments are still incorporating in their projects for the evolution of the hospital of the future, a growing number of interoperable medical devices in order to be able to implement closed cycle processes (monitoring, analysis, decision-making and reaction or application of a treatment), while at the same time they have been suffering in recent years an increasing number of successful malware attacks, demonstrating that the systems current detection systems are not very effective. A particularly problematic malware has been the so-called ransomware, which consists of infecting one of the devices on a network thanks to a vulnerability or human failure, and reaching the rest of the devices by means of a horizontal spread, usually based on vulnerabilities. System, after which it encrypts all the data on your hard drives and shared folders and requires an amount of money to provide the decryption key. To appreciate the potential danger posed by ransomware, suffice it to mention the attacks suffered by UK health service hospitals in 2017, which ended up having to shut down entire services, send patients to other hospitals and even postpone surgical interventions.

In this context of encryption traffic, solutions based on deep inspection are also not applicable, and most IDS-focused solutions that handle encryption traffic are based on in identifying certain basic patterns such as port scanning or brute force attacks [9]. There are proposals based on machine learning that use data calculated from a flow [10], and even offer imaginative proposals that use convolutional networks to treat the flow as if it were an image [11]. However, if we cannot access the payload because it is encrypted, a single stream does not provide enough information to get an accurate detection.

Finally, the third weak point of our interest is motivated by the proliferation of new malware, generally derived from existing versions to which characteristics such as the architecture on which they are executed or the model are changed encryption method [12]. IDS have difficulty in identifying these variations by working normally by examining the traffic by means of rules, which makes their early detection impossible. In the case of botnets and ransomware, both allow the generation of new versions easily. As proof of this, in [13] they conclude that nine of the ten most dangerous filtering botnets

are variations of the Zeus botnet. Similarly, in ransomware we find a great diversity of members of the same family, for example, Petya ransomware is derived from NotPetya, ExPetr or PetrWrap, among others.

Botnets and ransomware have in common that they generate network traffic following characteristic patterns. In the case of botnets, they usually have a command and control mechanism, Command and Control (C&C) whereby each infected device communicates with the attacker's computer periodically to receive orders. In the case of ransomware, its trafficking patterns stem from its desire to spread horizontally to maximize the damage and thereby increase the probability of payment of the ransom, from communicating with a central server for the obtaining the encryption keys and the traffic necessary for encrypting shared folders on the network of infected computers. These patterns can be interpreted as anomalies in normal network traffic.

1.1.1. Detection of Anomalies in Data Networks

An anomaly can be defined as a pattern that does not conform to the expected behavior or normal, which implies that it appears very infrequently. Precisely because it is based on the concept of normality, which in itself is not easy to define, the problem is far from being simple, there are mainly three large categories of anomalies [14]:

- Punctual. It is the simplest form of anomaly and is where most of the research in this field is focused. A sample of a group of data that is considered anomalous with respect to the rest is an example of this type.
- Contextual. An instance is abnormal in one context, but not in another. for example, a temperature of 5 degrees is abnormal or not depending on the season.
- Collective. A collection of instances can be considered an anomaly with respect to a set of data if, although each of the instances does not suppose an anomaly, the appearance of all of them as a collection s it is. A slight elongation of part of the wave of an electrocardiogram can be an example of this type of anomaly. The values belong to the normal range, but the sequence of values itself constitutes the anomaly.

The techniques for detecting anomalies based on machine learning act as classifiers capable of distinguishing abnormal and normal instances, it was possible to find approaches based on supervised, semi-supervised and unsupervised learning. Within these categories, it is to be expected that, if there is a properly labeled data set, the supervised approach will give the best results by having more complete information. However, this approach has as its main challenges the difficulty of obtaining a representative data set and the fact that anomalies are usually orders of magnitude less numerous than normal cases, with the consequent imbalance of the data set that complicates the classification task.

Anomalies in network traffic can belong to any of the three previous types. To give an example of each one, a specific anomaly could be directing a packet to a suspicious port; in the traffic between two devices that exchange small packets, a contextual anomaly will be the appearance of a large packet, since it will not be considered an anomaly. If the devices involved were others on the same network that would use that packet size; finally, in a context in which packets are emitted in bursts of a certain known duration, a change in that duration can be considered a collective anomaly. In this research, the aggregation of owes over time is used to be able to apply methods for detecting specific anomalies to contextual and collective anomalies.

1.1.2. Use of Network Flows in Anomaly Detection

Using the flows to detect these anomalies has multiple advantages, among which are the following: it is not necessary to access the payload of the packet, so it is applicable to encrypted traffic; reduces the volume of data to be analyzed by orders of magnitude, so it can be applied to environments with high-speed networks; finally, they respect the privacy of the user. The issue of privacy is crucial when it comes to obtaining permission from organizations to capture the massive amount of traffic that some of the newer machine learning algorithms require for their training. Working with owes makes it easier to ensure user anonymity and privacy, since administrators know that only packet headers will still be required. These reasons make them attractive for this research. Of course,

working with flow has drawbacks as well. The main one is the loss of information that involves losing the content of the packets and limiting oneself to obtaining aggregate information about the sequence of packets. However, this research aims to show that although a single flow may not be sufficient to extract complex information on the traffic pattern, a sequence of flows, properly treated, may contain enough information for a machine learning algorithm to differentiate abnormal from normal patterns. In this way, we comply with the restrictions of the proposed scenarios, by performing the detection without having to examine a high volume of data or worry about the fact that the data travels encrypted.

Detecting malware and attacks by analyzing network traffic remains a challenge for those responsible for monitoring network security and managing security incidents [11]. Although there are several well-known detection mechanisms to precisely separate malicious behaviors than normal, it is still extremely difficult to have efficient detection systems.

There are four main obstacles to a good detection of malware and attacks by analyzing network traffic. The first, that normal traffic is extremely complex, diverse and changing. Second, malicious actions are continually changing, adapting, migrating, and hiding like normal traffic.

Third, the amount of data to analyze is enormous, forcing analysts to lose data in favor of speed. And fourth, detection must occur in near real time to be of any use. For some years now, intrusion detection systems have incorporated intelligent paradigms such as machine learning techniques to solve these difficulties. Today there are also some proposals to implement Ensemble Learning or Ensembling algorithms, in order to combine multiple classifiers to achieve better detection precision. Ensemble Learning algorithms implement techniques to use, aggregate, and summarize information provided by several different detectors in a single final decision [24]. These allow security analysts to use serial weak detectors, vote to determine if a domain is malicious, and better decide the blocking action based on conflicting data. among other functionalities.

While there have been some good proposals for teaching techniques applied to network security [51], it is a topic that is currently under development. In particular, there are two aspects of the Ensembling algorithms that were not fully studied. One of them is the use in Ensemble Learning algorithms of threat intelligence data (Known also as TI), for example, VirusTotal [53]. And the second aspect is that there are no Ensemble Learning algorithms that work as a function of time in detection, that is, they take into account what happens on the network in a given time interval.

1.2. Proposal Aims and Objectives

The main objective of this proposal is to investigate how to apply machine learning methods to the detection of anomalies in data networks with restrictions. In the first case, the restriction is imposed by the impossibility of analyzing the payload of all packages by the volume of circulating traffic; and in a second case, the restriction consists of having to work with encrypted traffic and the short time for detection and mitigation. The hypothesis that this thesis raises after analyzing the proposals to classify network traffic based on existing flows in the literature, is that one flow by itself, without access to the payload of the packets, does not provide enough information; and it is proposed to study whether a context for this flow, formed by the flows previously received during a period of time, allows a more precise detection of anomalies in complex traffic patterns, in order to which will be necessary to use automatic learning methods for detecting anomalies, both classic and deep. To achieve this objective, a series of specific actions detailed below have been carried out:

1. Investigate and evaluate ensembling algorithms applicable to network security.
2. Develop and implement a method of detecting infected hosts, based on ensembling, that takes into account the detection results of different classifiers, using machine learning techniques and data from Threat Intelligence; and can work with time windows and detection over time.

1.3. Thesis Contribution

The contributions of this thesis are:

- The design of a methodology to detect infected hosts on the network using Ensemble Learning.
- The establishment of a procedure to test the methodology through experiments using real datasets and their results.

1.4. Proposal Scope

The working with data security is not that simple task, supposing that an important data has been transferred using a specific network. The data as to be secured against other connected process on the same network. From this concept many techniques suppose to hash the values then translate it into the correct form such in inscription and decryption case. From this importance our study aimed to be implemented in networks. In order to secure that transmitted data from attacks from other devices in the same network.

Other possible scope for our proposal is to use it in data connectivity network to work as data defender wall in data control layer for both the transmitter and the receiver.

1.5. Required Resources

The resource for our project divides into two main parts, Software and Hardware requirements:

1.5.1. Software Requirements

Tool	Specification
Python IDE	v. 3.8
Windows OS	Win10, 64-bit

Table 1. 1: Proposed Methodology Software Requirements.

1.5.2. Hardware Requirements

Resource	Specification
CPU	Intel Corei7 (8 CPUs) 10^{TH} Generation, ~2.3GHz
GPU	NVIDIA GeForce, 8GB
RAM	12GB
Hard Disk	SSD primary storage

Table 1. 2: Proposed Methodology Hardware Requirements.

Chapter Two: Literature Review

2.1. Network Flows

In order to facilitate network traffic analysis, packets are grouped into network traffic flows, or connections. A flow is defined as a set of IP packets that pass an observation point on the network during a certain time interval. All packages that belong to a certain flow have a set of common properties (packet attributes), known as "flow keys" which are: source IP address, destination IP address, source port, destination port and protocol. There are several protocols for the flow of network [34], including NetFlow, sFlow and IPFIX [15]. The flows are generated by different tools, including Argus [54], Zeek [44] and NFDUMP [47]. The difference between them is that Zeek generates owes when the capture ends, while Argus (like other network flow sensors) allows to report the flow every a certain amount of time, which configurable. This is called flow-report-time and it is important because it "cuts" the flow each time a time interval ends. Working with flow allows you to have less data and scale better. Although data is lost, it provides the ability to summarize the characteristics of the connection.

2.2. Threat Intelligence

It is called Threat Intelligence (TI) or threat intelligence to knowledge based on evidence, which includes context, mechanisms, indicators, implications and recommendations oriented to action on an existing threat or danger or emerging for assets. This knowledge can be used to make decisions related to the response that each organization gives to a given threat [42].

Today's threats evolve and become more complex at an accelerating rate and are continually confronted by many organizations. This is why Threat Intelligence has become a hot topic. In order to counter the increase in attacks, many organizations are

trying to incorporate sources of threat data into their network (better known as information feeds).

Beyond the relevance of Threat Intelligence information, a real understanding of it is important so that its use is appropriate and profitable. However, organizations lack that understanding and don't know what to do with all that extra data. This increases the burden on analysts who may not have the tools to decide what to prioritize and what to ignore [21].

In addition, threat intelligence is required to be actionable. In other words, it must be timely and arrive in a format that can be understood by whoever is consuming it. One way to achieve this is when Threat Intelligence information is easily integrated with all the security solutions already present in your environment. There are three categories of IT sources: internal, external, and community [5].

- Internal sources collect threat information within the organization, specifically from logs provided by some internal network services or applications (email log, alerts, incident response report, event logs, DNS logs, firewall logs, etc.) and the SIEM systems (Security Information and Event Management) that are implemented in them.
- External sources are organizations that provide threat information and have extensive data coverage. This information needs to be analyzed in each organization to determine its relevance (based on knowledge of the organization's services and the impact of threats on them). There are external sources that provide the data at no cost and others that are paid.
- Community sources are those that share threat information through a trusted channel between members with the same interest.

VirusTotal is an information feed that provides the results of scanning for suspicious files and URLs. VirusTotal inspects these items with a host of antivirus and domain and URL blocklist services. In addition, it uses a set of tools to extract signals from the analyzed content [53].

The analysis of suspicious files and URLs can be done through the WEB page, the browser extensions or the API that VirusTotal provides. By submitting a file or URL, the results are shared with the requester, as well as among examining partners, that is, other VirusTotal users. The latter use the results to improve their own systems. Consequently, when sending files, URLs, domains, etc. At VirusTotal, you contribute to the VirusTotal Community and global Internet security.

VirusTotal can be useful for detecting malicious content and also for identifying false positives: normal and harmless items detected as malicious by one or more antivirus engines. The API (Application Programming Interface) allows access to the information generated by VirusTotal without the need to use the interface of the HTML website.

It has two versions: a public API and a private API. These differ in the maximum number of requests that can be made per minute and the priority these queries have for the VirusTotal engine. Currently the public API allows up to 4 queries per minute and provides minimal access priority. Whereas in the private API, the request rate and the total number of allowed queries are still limited only by the user's terms of service.

In particular, VirusTotal provides the following information for an IP address or for a domain [52]:

- Autonomous System (AS) and location country for IP addresses. The country information is given in the country code.
- Passive DNS Replication Information - All IP domain name mappings that VirusTotal has seen for the item it queried for.
- Whois searches: registration information for the resource for which it is consulted, such as the domain name, the IP block or the autonomous system.
- Observed subdomains: domains viewed hierarchically under another domain stored in VirusTotal.
- Sibling domains: domains at the same hierarchical level as the domain being analyzed.
- URLs: the last URLs seen under the domain or IP address that is being analyzed.

- Downloaded files: last files that have been downloaded from the URLs located in the domain or the IP address in analysis.
- Communication files: more recent files that, through their execution in an isolated virtual environment, perform some type of communication with the IP address or domain in question. .
- File references: VirusTotal inspects the strings contained in the files sent to the service and applies certain regular expressions to them to identify domains and IP addresses. It then registers the files that have made reference to the domain or the IP address in question.

2.3. Ensembling Techniques

Ensemble Learning is a paradigm of Machine Learning where multiple agents (called base learners or base algorithms) are combined and trained to solve the same problem. Unlike classical Machine Learning techniques which try to learn a model (hypothesis) from training data, Ensemble Learning techniques try to build a set of models and combine them to use them in prediction [56].

Ensemble Learning can be: Homogeneous: when it uses a single learning algorithm, that is, it combines base algorithms of the same type (homogeneous). Heterogeneous: when it uses multiple learning algorithms, that is, it combines base algorithms of different types (heterogeneous).

The Ensemble Learning methods imply an improvement in generalizability and the predictive power of learning. Often the level of precision of the base algorithms is slightly higher than chance because they have high bias or too much variation. Ensemble Learning tries to reduce both variables by combining the base learners as shown in Figure 2.1. The three most popular methods for combining predictions from different models are [56]:

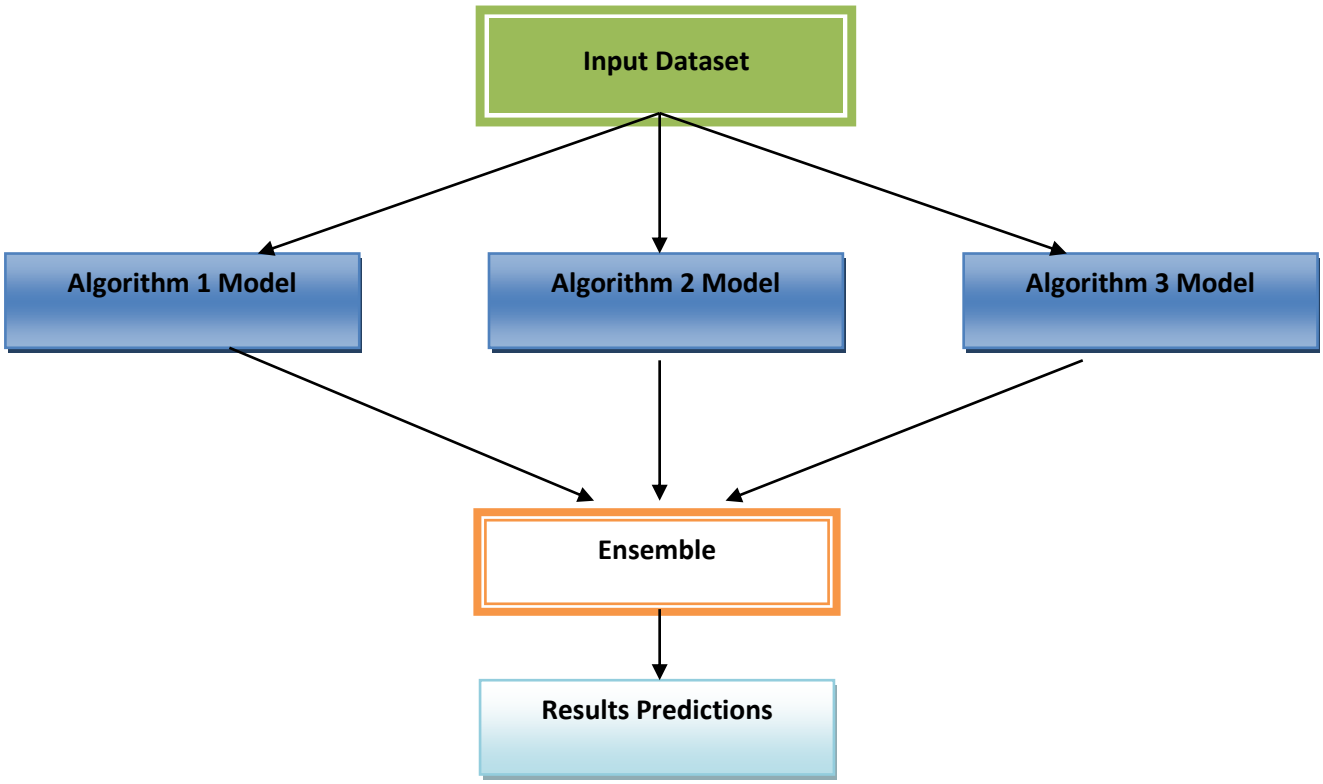


Figure 2. 1: Ensemble learning general architecture. Predictions are the result of applying some teaching technique to combine different models.

- Bagging: build multiple models (typically of the same type) from different subsamples of the training data set.
- Boosting: build multiple models (typically of the same type), each of which learns to correct the prediction errors of the previous model in the sequence of models.
- Voting: build multiple models (typically of different types) and use simple statistics (such as calculating the mean) to combine predictions.
- Another Ensemble Learning technique, which some authors also consider popular, is Stacking or Stacked Generalization. It uses a meta-learning algorithm to learn how to best combine the predictions of two or more Machine Learning algorithms.

2.3.1. Bagging Ensemble

Bagging or Boostrap Aggregation takes multiple samples from the training data set (with replacement) and trains a model for each sample. Figure 2.2 shows the architecture of this technique. It uses what is known as a boostrap sample. A bootstrap sample (nonparametric) is a random selection of several elements from the data set with replacement. That is, a bootstrap sample can contain multiple copies of one of the original data [4].

The final prediction is obtained by averaging the predictions of all the sub-models, in the regression problems. Whereas in classification problems, the final prediction can be obtained both by averaging the predictions and by using probabilities based on the percentages of the different classes.

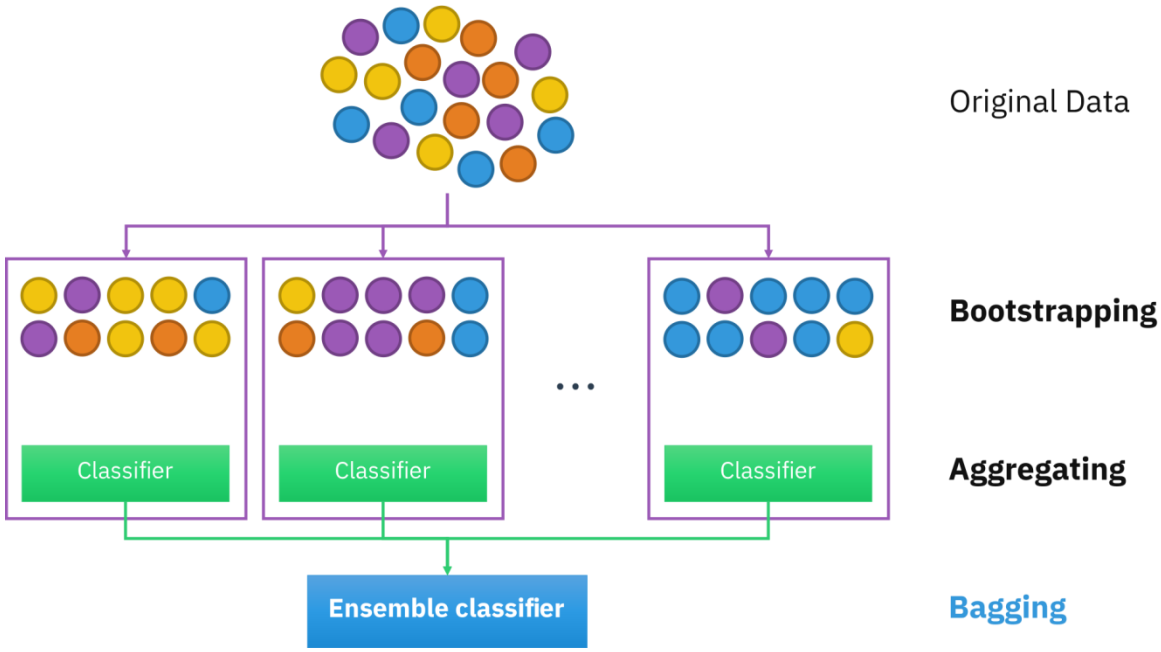


Figure 2. 2: Architecture of the Ensemble Learning Bagging technique.

2.3.2. Boosting Ensemble

Boosting creates a sequence of models that tries to correct the errors of the models before them in the previous sequence, as shown in Figure 2.3. Once created, the models make predictions that can be weighted for their demonstrated accuracy, and the results are combined to create a final prediction [4].

In a first step, the algorithm is trained on the entire data set. Subsequent models are built by fitting the residuals from the previous algorithm. This is done by giving more weight to the observations that the previous model incorrectly predicted. It is based on the creation of a series of weak algorithms, each of which may not be appropriate for the entire data set, but is appropriate for a part of it. Therefore, each model increases the performance of the whole.

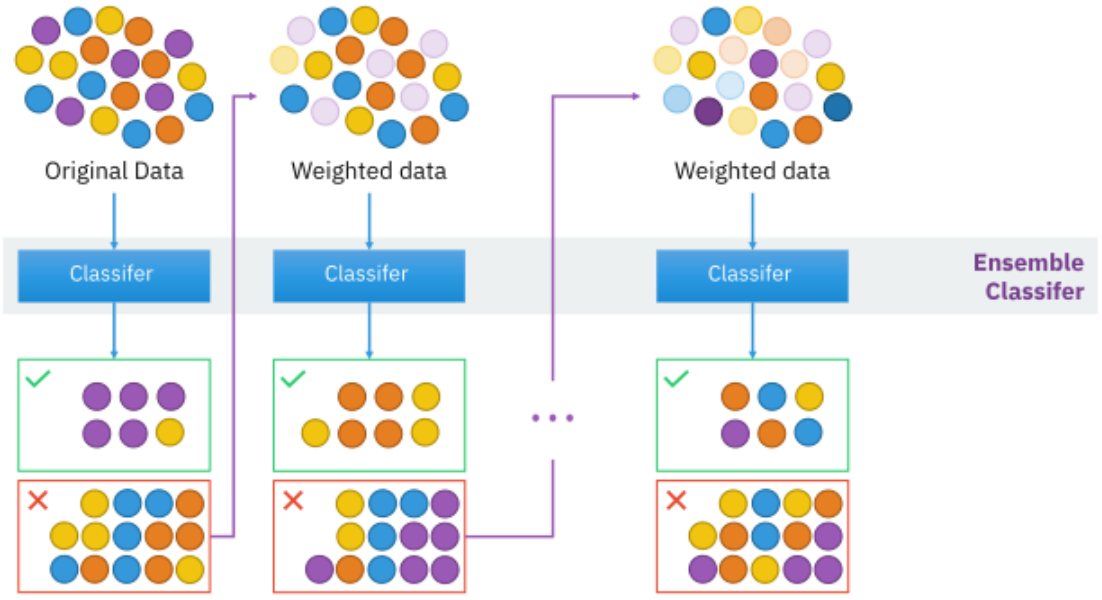


Figure 2. 3: An illustration presenting the intuition behind the boosting algorithm, consisting of the parallel learners and weighted dataset.

2.3.3. Voting Ensemble

Voting is one of the simplest ways to combine the predictions of multiple machine learning algorithms, its operation can be observed in Figure 2.4. It works by creating two or more independent (heterogeneous) models from your training data set. A voting method is then used to make predictions [56]. Within this technique there are variants, such as Majority Voting or Hard Voting, Soft Voting and Weighted Voting.

In Majority Voting, several models are trained with the same data. When predicting, a prediction is obtained from each model. Each model will have a vote associated with it. Then the final prediction will be determined by what the largest of the models vote [26].

In Soft Voting, soft vote is used. "In this variant, more importance is given to the results in which some model is very sure. That is, when the probability of the prediction is at very close to 1, more weight is given to the prediction of this model [26].

Weighted Voting is used when individual classifiers have uneven performance. Giving more power to the strongest classifiers in voting, a weighted voting is carried out. Sub model predictions can be weighted, but it is difficult to specify classifier weights manually or even heuristically. The more advanced methods learn how to better weight the predictions of the sub-models.

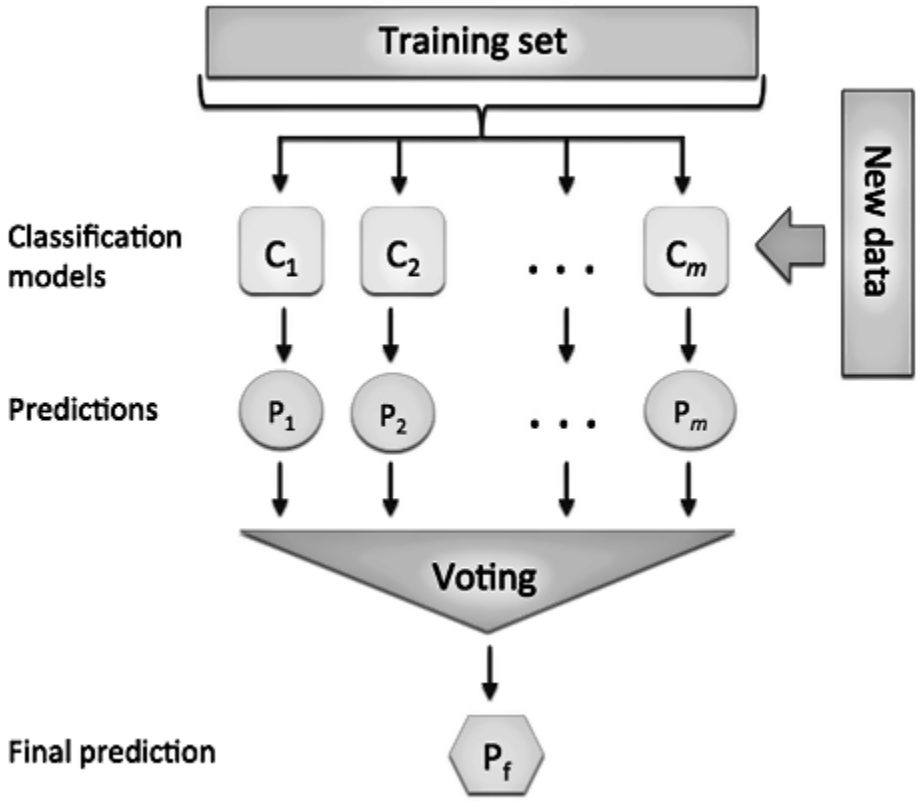


Figure 2. 4: Majority Voting Techniques.

2.3.4. Stacking Ensemble

Stacking or Stacking consists of training a model to perform the aggregation or combination of the predictions of all the sub-models, instead of doing it using trivial functions (such as Hard Voting). In a first step, the sample is divided into a subsample for training and a subsample for testing. Then a set of base algorithms is trained with the training sample. And the resulting models are evaluated using the test sample. Each of the sub-models predicts a different value and finally the ensemble learner (also called meta-learner) takes these predictions as inputs and makes the final prediction. Its architecture can be seen in Figure 2.5. When Stacking is implemented, the ensembling can be homogeneous or heterogeneous. As described earlier in this section, homogeneous ensembling combines base algorithms of the same type while heterogeneous ensembling combines base algorithms of different types.

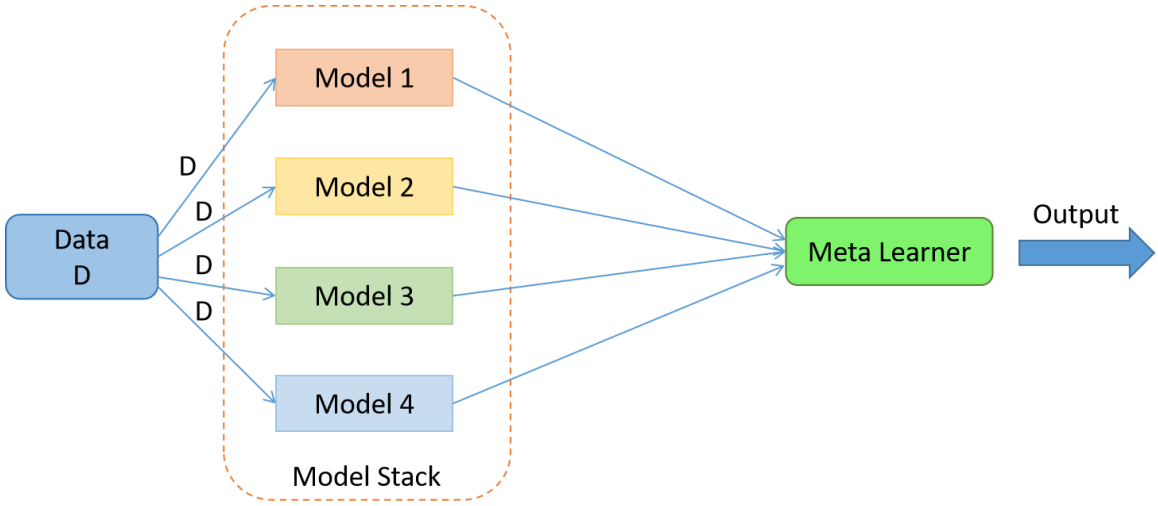


Figure 2. 5: Stacking Ensembling.

2.4. Slips

Slips (Stratosphere Linux IPS) is a modular intrusion prevention system developed in Python. It is based on machine learning techniques to detect malicious behavior in network traffic [31]. Slips was designed to target targeted attacks, such as Command and Control channel detection, to provide a good view for the security analyst From network traffic, create profiles for each address on IP, and then divide the traffic into time windows. For each time window, Slips extracts characteristics of the traffic and then analyzes them in different ways in order to detect malicious behavior [33].

At the time of writing this thesis, the last published version of Slips is 0.6.8. It detects horizontal and vertical port scans, as well as the existence of various command and control connections. A slip is free software and is available on the Stratosphere page [32]. Slips can be run on Debian and MacOS based Linux systems (10.9.5, 10.10.x, 10.12.x). In its current version, Slips provides the facility to run inside a Docker container [40], to analyze files from network streams or pcap files. However, to perform network traffic analysis in real time, the only option is to perform the traditional installation of the tool.

2.4.1. Slips Architecture

Slips works at the network flow level, rather than packet inspection, obtaining a high-level view of behaviors. Slips create traffic profiles for each IP address that appears in the traffic. A profile contains the behavior complete of each one of them. Each profile is divided into time windows. Each time window is 1 hour long by default (can be set by con guration) and contains dozens of calculated functions for all connections starting in that time window. Detections are made in each time window, allowing the profile to be marked as not infected in the next time window.

Slips can read streams from different sources such as pcap files or outputs from Zeek, Bro, Nfdump, among others. Once it is done, the data is processed and inserted into the profile of each source IP address. For each IP address analyzed, Slips creates this Pro le structure that represents the pro le for that address and contains three types of logs: Time Windows Files, Timeline le and Pro le File. In addition to the profile information, Slips creates some files with information about the entire capture, such as Blocked.txt. This file has information about all the IP addresses that were detected and blocked.

2.4.2. Slips Main Modules

Slips have modules, which are files written in Python. They allow any developer to expand its functionality [27]. They process and analyze data, perform additional detections, and store data in the Redis database for other modules to consume. Currently, Slips has the following modules:

1. ASN module: allows you to load and find the Autonomous System Number (ASN) of each IP.
2. Geoip IP module: its function is to find the country and geolocation information of each IP.
3. Https module: allows you to train or test a RandomForest classifier to detect malicious HTTPS streams.

4. Port Scan Detector module: its function is to detect horizontal and vertical port scans.
5. Threat Intelligence module: used to check if each IP address is on a list of malicious IPs.
6. Timeline module: allows you to create a timeline of what happened on the network, based on all the flows and types of data available.
7. VirusTotal module: its function is to look up the IP address in VirusTotal.
8. Kalipso [1]: it is the graphical user interface to show the traffic analyzed by Slips.

The core of Slips is not only constituted by the machine learning algorithms, but also by the behavioral models that are used to describe the flows as a function of duration, size and periodicity. Thereof, this is very important because the models are still selected to maximize detection.

Finally, a characteristic that makes Slips attractive is that it implements an API, from which a new detection algorithm can be easily incorporated that any developer implements, thus giving the possibility that the tool grows from the contributions of the community.

2.5. Related Studies

Network intrusion detection is an important research area, as cyberattacks are increasing daily [11]. There are numerous studies to propose approaches to detect them. However, as cyberattacks become more complex, existing approaches fail to tackle the problem effectively.

That is why the detection of intrusions in the network continues to be a challenge in decision-making, which can be addressed through the application of classification algorithms [22]. These algorithms use machine learning techniques to detect network attacks and malware. This offers the following advantages:

- The ability of machine learning to generalize and thus detect new types of intrusions.
- Attack signatures can be automatically extracted from tagged traffic data.

- The ability to adapt to new attacks.

In order to benefit from multiple different classifiers, and exploit their strengths, the use of ensembling algorithms [56] [55] arises, which combine the results of the individual classifiers into a final result to achieve a best result. From this combination of results, the performance of individual algorithms can be improved. Some studies have shown that the application of the Ensemble Learning paradigm in intrusion detection systems can be versatile and undoubtedly improve the accuracy of the prediction and the speed of detection. [12] [13]. In particular, the highest speed in the detection using ensembling can be achieved from the use of parallel architectures.

On the other hand, cyberattacks have different types of characteristics: general, trafficking or associated with the connection, content or associated with data. The selection of characteristics is essential, so it is important to continue investigating the respect. It is also essential to evaluate which base classifiers to use and how they should be combined in order to design architectures where multiple classifiers collaborate with each other rather than compete.

Proposals based on the stacking technique have also been developed to detect network intrusions (Probe, DoS, UR2 and R2L) [38]. In this work the models are generated using samples from the random selection of characteristics of the dataset. It is proposed to select the best models according to a nest criterion (such as precision, rate of true positives, among others) and combine them with Stacking as an Ensemble Learning technique.

Other works focus on botnet detection based on the classification of network traffic flows [18] [35]. In [18] they propose to carry out the detection in two stages. In the first stage, they apply a clustering algorithm to generate clusters that group network flows with similar characteristics. And in the second stage, classification algorithms are applied to each cluster to separate botnet flows and normal flows.

Given the instability of the clustering algorithms, these methods have shortcomings, which try to be improved with some variants as proposed in [35] through the use of link-based algorithms to group the network flows in stage 1 instead of clustering algorithms.

Beyond the mechanisms and tools, to implement an effective defense, the organization needs to have information about possible adversaries, as well as such as its techniques, tactics, and procedures. This so-called threat intelligence helps the organization better understand its threat profile [14]. Threat Intelligence (IT) feeds or threat intelligence sources allow organizations to obtain indicators that are used by their firewalls and their systems.

Intrusion detection system for a timely reaction to emerging threats. Intelligence sources are usually made up of simple indicators. They can provide information on suspicious domains, lists of known malware hashes, or IP addresses associated with malicious activity, among other things. With the information provided by these intelligence sources, organizations they often choose to blacklist communications and connection requests that originate from malicious sources, for example [28]).

While intelligence sources can be easy to understand and use, they are not a complete solution. They do not provide context or prioritize threats, so it is necessary to have procedures and mechanisms to extract value from them and use them properly. In addition, although they are widely used in the industry as a useful tool to mitigate attacks, there are studies that affirm that their quality is not as high as expected, nor is their specificity and its completeness [50] [9] [29].

Consequently, it can be thought that a good use of the information provided by intelligence sources is to integrate and combine it with the mechanisms for detecting malware and attacks on the network that the organization has. However, none of the proposals previously described in this chapter includes the information provided by IT sources in the classification process.

In this context, it is proposed to contribute to the cybersecurity field, and to the detection of infected hosts in particular, taking advantage of ensembling techniques [10] [33], including information on the IT sources in the detection process.

Chapter Three: Ensembling Techniques to Discover Hosts Infected on the Network

The proposed methodology aims to improve the process of detecting infected hosts on the network. The hosts to analyze are those that initiate network connections. The detection is done through the implementation of Ensemble Learning. The decisions to be made during the process are based on different data provided by Slips [31].

It works with all the flows provided by a device that generates flows for the target network. To determine if a host is infected within a time window, the following information is considered:

- The different predictions for each network flow, one for each classifier.
- The set of malicious behavior alerts associated with the given IP (originating from this IP).
- The data provided by different Threat Intelligence sources that inform whether the destinations of the analyzed flows are malicious or show signs of being (with some percentage of confidence).

Based on the above, it is proposed to apply Ensemble Learning to make different decisions. First, to determine if each flow is malicious or normal. Second, to determine if the set of flows that go from an origin to a destination are part of an infection. And third, to decide if each source IP address is infected or not. The work has been carried out in phases, in order to modularize the analysis, carry out experimental tests for each one, obtain results for a part of the problem in particular and be able to adjust the solution based on it.

3.1. Phase 1: Ensemble Learning to Classify Network Flow

The objective of this first phase is to assign a label for each network flow. The value will be malicious or normal. This value is the result of applying Ensemble Learning, on the predictions given by the different classificator of Slips, as shown in each network flow

has a set of n labels from the predictions of the n classifiers in Slips operation. These predictions will be combined in such a way as to obtain a single decision for each flow.

The labels assigned by the Slips classifiers are the result of applying base machine learning algorithms to classify flow. To combine those labels in order to make a decision is that Ensemble Learning is applied. To determine the most appropriate Ensemble Learning technique to implement in this phase, different experiments were carried out and the results were evaluated, described in the next chapter. From them it was decided to use Weighted Voting in this instance.

As a result of this phase there is then a label for flow: malicious (for malicious) or normal. The output of this phase is a dataset with the same fields as the incoming dataset, replacing the n labels with a single label, resulting from the ensembling applied.

3.2. Phase 2: Ensemble Learning to Classify Sets of Flows from Source to Destination

The objective of this second phase is to decide the label assignment for each set of network flows that represents all the connections between an origin and a destination. Said label will be malicious if the set of owes is part of an infection, and normal otherwise. As a result of this phase there is a set network streams labeled.

Ensembling is carried out as follows:

- For the set of flows that go from a source IP (SrcAddr) to a destination IP (DstAddr), the established TCP connections, the non-established TCP connections, the established UDP connections and the non-established UDP connections are analyzed (considered UDP connections established to the UDP streams that received a response and UDP connections not established to those that did not received it).
- For each of the groups (TCP connections established, TCP connections not established, UDP connections established and UDP connections not established), the percentage and quantity of flows labeled as malware in Phase 1 is calculated. then 2 values: Flow $_{\text{Percentage}}$ = Malware and Amount $_{\text{Flows}}$ = Malware.

In this phase there are 2 levels of decision:

- First level of decision: you have to label each group of flow's (TCP Connections Established, TCP Connections Not Established, UDP Connections Established, and UDP Connections Not Established) as malicious or non-malicious.
- Second level of decision: its purpose is to label the total set of network flows that represents all the connections between the origin and the destination, as malicious or non-malicious. It is the final decision of Phase 2.

As a result of this first-level decision process, an intermediate dataset is built, which includes for each source IP-destination IP pair, the label for each group. The labels for each group are:

- TCPELabel: label assigned to the group of established TCP flows that go from the source IP address SrcAddr to the destination IP address DstAddr.
- TCPNELabel: label assigned to the group of non-established TCP flows that go from the source IP address SrcAddr to the destination IP address DstAddr.
- UDPELabel: label assigned to the group of established UDP flows that go from the source IP address SrcAddr to the destination IP address DstAddr.
- UDPNELabel: label assigned to the group of unestablished UDP flows that go from the source IP address SrcAddr to the destination IP address DstAddr.

The decision to take into account the amount of malicious flows by type of connection, in addition to the percentage, is to rule out cases where a single flow can represent 100% of the connections of that type, and treat only from an attempted attack.

3.3. Second Level of Decision

At this level, the label must be decided for the set of all the flow that go from origin to destination. For this, an ensembling technique similar to stacking is proposed. We are considered to have 4 labels for each set of owes. Each of them is obtained by analyzing a subset of different characteristics of the set of flow going from an origin to a destination:

- TCPELabel: is the label that is assigned to a set of flows that go from a source to a destination based on the established TCP flows. It takes into account the decision made at the first level regarding this group of owes.
- TCPNELabel: is the label that is assigned to a set of flow that go from an origin to a destination depending on the TCP streams not established. It takes into account the decision made at the first level regarding this group of owes.
- UDPELabel: is the label that is assigned to a set of flows that go from an origin to a destination based on the established UDP flows. It takes into account the decision made at the first level regarding this group of owes.
- UDPNELabel: is the label that is assigned to a set of flows that go from an origin to a destination based on the non-established UDP flows. It takes into account the decision made at the first level regarding this group of owes.

To decide the label for the entire set of owes, these 4 labels are combined. In this way, if any of the groups is classified as malicious, then the whole group will be classified as malicious. The output of this phase is a dataset resulting from adding, to the intermediate dataset, a column with the prediction of Phase 2, called PredictLabel. This column indicates whether the set of flows exchanged between the source IP SrcAddr and the destination IP DstAddr are part of an infection. In this case the value of PredictLabel be malicious, and normal otherwise.

3.4. Phase 3: Ensemble Learning to Classify Hosts

In this phase the objective is to decide if each source IP is infected or not. For this, ensembling is applied from the information related to the destinations with which that host connects. This information includes the Phase 2 prediction and Threat Intelligence (TI) data.

For each destination IP address, the following is taken into account: The result of combining the information provided by different Threat Intelligence (TI) sources, such as VirusTotal. The Phase 2 Ensemble Learning decision, which tells us if the connections from the source to that destination are part of an infection.

It is necessary to consult the Threat Intelligence information for each destination IP address of the flows to be analyzed. For each Threat Intelligence source (feed), the criteria to be applied to use the data it provides are defined. This criterion is in accordance with the level of trust you have for that IT source.

This is done for all IT sources considered to be included. In this case, the IT modules that Slips implements are used. The logic defined here allows new IT sources to be easily brought into the process. In this model, VirusTotal is incorporated as a feed, Slips features a

module called VirusTotal Module that communicates with said feed. From the information obtained from it, the module calculates the url ratio, download ratio, communicating ratio referrer ratio. These 4 values are used by this phase in the classification process.

For each IP, the VirusTtotal API returns data on 4 categories: URLs that resolved to the IP, samples (files) downloaded from the IP, samples (files) containing the given IP, and samples (programs) that they contact the IP.

The data structure is the same for the 4 categories. For each sample in a category, VirusTotal queries the antivirus engines and counts how many of them find the malicious sample. The answer has two fields for each category. These are the "detected category" field, which contains a list of samples that were found malicious by at least one engine, and the "undetected category" field, which contains all samples that none of the engines found or malicious. The answer has two scoring fields (detected and not detected) for each of the 4 categories.

From this response from the VirusTotal API, the VirusTotal Slips module calculates the ratio of each category. To calculate the proportion of a category, the global number of detections is calculated (sum of all positive detections (detected) from the list of all samples) and the global number of tests (sum of all positive and non-positive detections (detected and undetected) from the list of all samples). For the IP consulted, we then have the total of positive detections and total of tests for each category.

Chapter Four: Adopted Dataset

Datasets from the Stratosphere Laboratory [48] and "adhoc" datasets were used to carry out the experiments. The latter were generated in the different phases, to be used for data processing (intermediate) or as an interface with the next phase (resulting). The Stratosphere datasets are created from actual captures of normal, malicious and mixed traffic. The latter category includes captures of both normal and malicious network traffic simultaneously.

The malware captures were carried out within the framework of the Malware Capture Facility Project [30], in which the threat landscape is continuously monitored to detect new emerging threats, retrieve malicious samples, and execute them on its premises in order to capture the threat scenarios are created both infected and uninfected machines participate, and network traffic is captured using different tools. From the captures, datasets are generated that are published for the community. Each scenario has an associated distinguishing identifier within the Stratosphere dataset repository.

The performance of machine learning algorithms must be verified with real data. Especially in cybersecurity, it is really important to have data representative of the network traffic, which includes normal activity and malicious activity, which includes different types of attacks and the different phases of them.

To do a good check, we need three types of traffic: malware, normal and "background". Malware traffic includes everything you want to detect, especially command and control connections. "Normal traffic is very important to discover the real performance of our algorithms by calculating false positives and true negatives. Background "traffic is non-malicious traffic that is transmitted at the same time as the malicious traffic.

The same is necessary to saturate the algorithms, check their memory performance and speed, and also test if the algorithm is confused with the data. Mixed shots provide a realistic scenario where a machine is not at infected, then it gets infected and after a while it stops being infected. This type of scenario makes it easy to test machine learning

algorithms and models. To carry out the training and testing, the first dataset provided by Stratosphere [48] was used.

- Dir: sense of connection.
- Dport: destination port.
- DstAddr: destination IP address.
- Dur: duration of the connection.
- Proto: transport layer protocol used by the connection.
- Sport: port of origin.
- SrcAddr: source IP address.
- SrcBytes: number of bytes sent to the source.
- SrcPkts: number of packets sent to the origin.
- StartTime: timestamp corresponding to the start of the connection.
- State: connection status. The possible values for this field are detailed in [43].
- TotBytes: number of bytes transmitted through that connection.
- TotPkts: number of packets transmitted through that connection.
- dTos: type of service of the destination.
- sTos: type of service of the origin.
- Label: label indicating malware or normal.

Chapter Five: Adopted Methodology

The main part in our methodology is to performed classification on the adopted dataset in order to estimate the following algorithms behavior on our adopted pipeline firewall methodology:

- Logistic Regression (LR)
- Random Forest (RF)
- Naive Bayes (NB)
- K Nearest Neighbor (KNN)
- Decision Tree (DT)

Each one of these algorithms will extract a classification model with classification accuracy that measures its validity on such classification case. On the other hand, the using of these algorithms will help us in apply the ensembling model in order to determine the best classification algorithm between these algorithms.

The pipeline will work with the following pseudo code:

PROCESURE PIPELINE (PROCESS P):

```
1   LR= Logistic Regression (P)
2   RF= Random Forest (P)
3   NB= Naive Bayes (P)
4   KNN= K Nearest Neighbor (P)
5   DT= Decision Tree (P)
6   ENs= ensembling (LR,RF,NB,KNN,DT)
7   IF LR>60 AND RF>60 AND NB>60 AND KNN>60 AND DT>60 THEN
8         BEST_ALGORTHIM= ENs[0]
9         FLAG_PIPELINE=FALSE (DON'T ALLOW PASS)
10        LOG ('detect intrusion using', BEST_ALGORTHIM)
11        LOG ('with accuracy', ENs[1])
12        LOG ('at time', Time_Now())
13        LOG ('process details', P.info())
14  ELSE
15        FLAG_PIPELINE=TRUE (ALLOW PASS)
```

Figure 5. 1: Adopted Methodology Pseudo Code.

From the pseudo code above we can see that the using of ensemble is to determine the best algorithm to adopt it accuracy in case all of algorithms agree that a process is 60 and above of percentage that it is an intrusion.

The process of build the system will be divided into two main sections, the first one is to train the algorithms one by one on the dataset where each algorithm needs training subset and testing subset. The training subset is used in order to train the algorithm functions on the data while the testing is used to validate the behavior of the algorithm during the training phase. Our dataset has been divided 80% training and 20% testing.

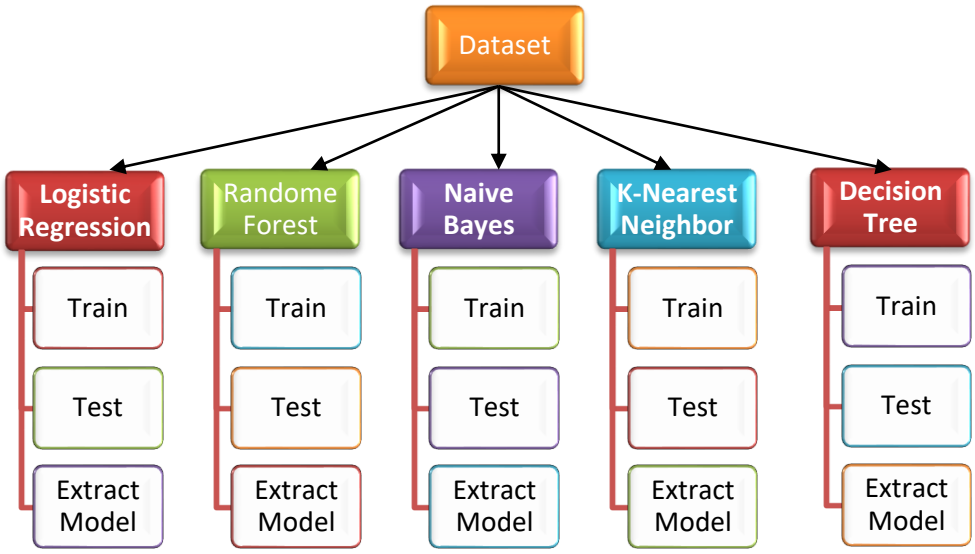


Figure 5. 2: Algorithms Training and Testing Processes.

While the 2nd section is to use the extracted models in real case with the ensembling model.

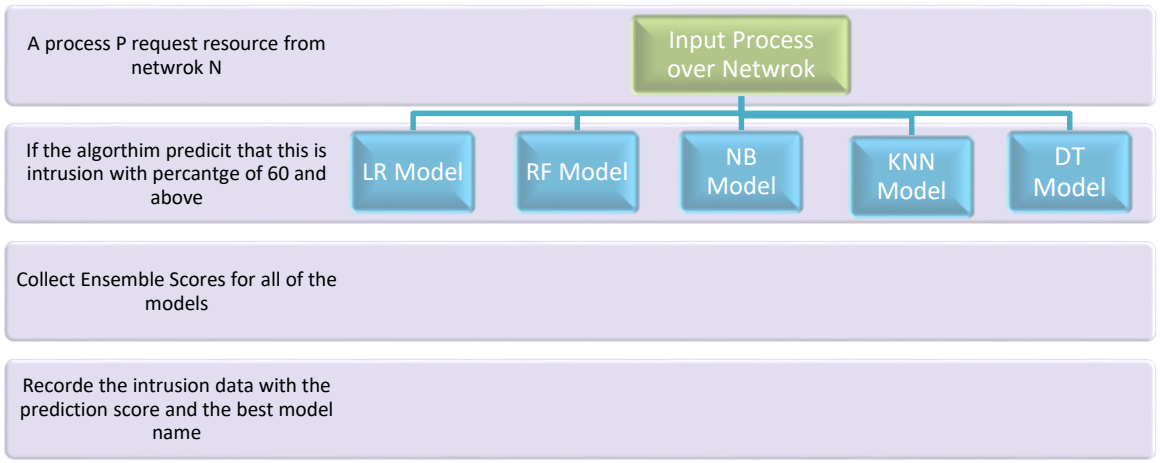


Figure 5. 3: Methodology Pipeline with Ensembling Testing.

Chapter Six: Result and Discussion

In order to validate the nest model, experiments were carried out that allow testing its different phases with real datasets, obtaining conclusions from the results, and adjusting the classification criteria for each phase. Experiments are the core of this work. Based on its results, the advantages of applying ensembling are analyzed in order to improve the detection of infected hosts. This is done as a stage prior to the implementation of a module that integrates the 3 phases.

In our model, each phase consists of two stages: a training stage and a testing stage (that is, results verification tests). In the training stage, the designed algorithms are trained in order to establish the best model for each of the phases. In each phase there are different variables that are part of the classification criteria, on whose values the results depend.

To establish them, the training is carried out, using different values, and the results are analyzed. To determine the best model, metrics are used. They can be calculated from the resulting confusion matrix. A confusion matrix is a representation of the performance of the classification models [39]. The matrix shows the number of cases classified correctly and incorrectly, compared to real labels, known as Ground Truth).

One of the advantages of using the confusion matrix as an evaluation tool is that it allows a detailed analysis from which different metrics are obtained. From this set of metrics, which should be chosen when used for the comparison of the models.

The performance of a binary classifier is summarized in a confusion matrix. The same cross-tabulates predicted (prediction) and observed (known truth or Ground Truth) cases in four options:

1. True Positive (TP): a positive label is correctly predicted.
2. True negative (TN): a negative label is correctly predicted.
3. False positive (FP): a positive label is predicted, and it was false.
4. False negative (FN): a negative label is predicted, and it was true.

The metrics used, which are obtained from each confusion matrix, are:

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

$$F1\ Score = \frac{2 * TP}{2TP + FP + FN}$$

$$\text{False positive rate (FPR)} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

$$\text{True positive rate (TPR)} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

In the testing stage, the chosen model is validated, with another data sample. The classifiers implemented in this thesis are binary classifiers. In each phase the criteria is the following:

- In Phase 1 the labels malicious and normal are used. In it, a detection is considered positive when an flow is classified as malicious.
- In Phase 2 the labels malicious and normal are used. At this stage, a detection is considered positive when the set of flows that go from an origin to a destination is classified as malicious.
- In Phase 3 the labels infected and normal are used. In it, a detection is considered positive when a source IP is classified as infected.

The following sections describe for each phase: the experiments carried out, detailing their objective, the training and testing stages, and the results obtained.

6.1. Phase 1 Testing: Malicious or Normal Based on Flow itself

As the first stage of this project, performance tests of the Ensemble Learning algorithms are performed to detect malicious flows, and its accuracy was compared with that of a set of core learning algorithms. To carry out the tests it is a mixed data set with labels corresponding to normal traffic and malicious traffic, from a botnet known as Rbot. The following ML algorithms are tested: LR, NB, RF, KNN and DT [3]. And the teaching techniques used were: Hard Voting or Majority Voting, Soft Voting (using the sum of the predicted probabilities), Weighted Voting or Weighted Voting, Bagging and Boosting.

Tests are implemented with the Scikit-learn library [16] and Accuracy or precision, obtained by applying the cross val score function to the model, is used as a metric. This function performs a simple division of the dataset data into a subset for training and a subset for testing [10].

Logistic Regression: **Accuracy 99,45%**

Random Forest: **Accuracy 99,999%**

Naive Bayes: **Accuracy 98,989%**

K-Nearest Neighbor: **Accuracy 99,999%**

Decision Tree: **Accuracy 99,99%**

The results above shows the precision obtained for the following automatic learning algorithms: Logistic Regression, Random Forest, Naive Bayes, K Nearest Neighbor and Decision Tree. These values are then compared with those resulting from applying different Ensemble Learning techniques to combine them.

Majority Voting	**99,731%**
Soft Voting	**99,753%**
Weighted Voting (LR=1, RF=3 y NB=1)	**99,993%**
Weighted Voting (LR=1, RF=3 y NB=2)	**99,991%**
Weighted Voting (LR=2, RF=3 y NB=1)	**99,988%**

Results of the Experiments using Logistic Regression, Random Forest and Naive Bayes as base algorithms and applying different variants of voting techniques as Ensemble Learning. The first is shown by the value of Accuracy when applying Majority Voting. The second sample shows the Accuracy value when applying Soft Voting. The following show the Accuracy value for Weighted Voting with different weights assigned to Logistic Regression, Random Forest, and Naive Bayes. These correspond to the weights for which the best Accuracy values were obtained

From the results above we can conclude that applying Majority Voting and Soft Voting obtain similar results. They both improve Logistic Regression and Naive Bayes but not Random Forest. This happens because, when Random Forest is combined that gives good results with two other algorithms that are worse, when applying Majority Voting goes wrong. While applying Weighted Voting there are no improvements for Random Forest, although it does show improvements compared to the other proven voting techniques.

However the precision is closer to Random Forest without teaching techniques. The best results in the classification are obtained by giving the highest weight to Random Forest and the least weight to the other two algorithms.

Weighted Voting (LR=1, KNN=2 y NB=1)	**99.975%**
Weighted Voting (LR=1, KNN=3 y NB=1)	**99.975%**
Weighted Voting (LR=2, KNN=3 y NB=1)	**99.974%**

Results of the experiments using Logistic Regression, K-Nearest Neighbor and Naive Bayes as base algorithms, and applying Weighted Voting as the Ensembling technique. In these experiments, the Random Forest algorithm was replaced by K Nearest Neighbor.

From the above results experiments are done with Weighted Voting replacing Random Forest by K Nearest Neighbor and Decision Tree.

Logistic Regression, Decision Tree, and Naive Bayes are shown in the results below. In both cases, improvements can be observed when applying Ensemble Learning compared to not applying it. Although the improvements when applying Weighted Voting are not significant, there are improvements for all the algorithms involved.

Weighted Voting (LR=1, DT=3 y NB=1)	**99.993%**
Weighted Voting (LR=1, DT=3 y NB=2)	**99.991%**
Weighted Voting (LR=2, DT=3 y NB=1)	**99.993%**

Results of the experiments using Logistic Regression, Decision Tree and Naive Bayes as base algorithms and applying Weighted Voting as an Ensembling technique. In these experiments, the Random Forest algorithm was replaced by Decision Tree.

Applying Bagging results in minor improvements for Random Forest, KNN, and DT, and no improvement for Logistic Regression and Naive Bayes. The results are shown below in table 6.1.

Algorithm	With Bagging	Without Bagging
LR	99.225%	99.125%
RF	99.15%	99.125%
NB	97.97%	97.75%
KNN	99.85%	98.99%
DT	96.10%	95.81%

Table 6. 1: Experiments using Logistic Regression, Random Forest, Naive Bayes, K Nearest Neighbor and Decision Tree as base algorithms and applying Bagging as Ensembling technique.

6.2. Phase 1 Testing

The first stage of testing allows carrying out the first experiments with automatic learning algorithms, in particular from Ensemble Learning, with the aim of detecting malicious owes. With these results and in order to advance in the process, other experiments are carried out with a larger dataset. It is built from the adopted dataset. This allows for a greater quantity and heterogeneity of data, in order to obtain better results. This dataset constitutes the input for the experiments of the whole ensembling process implemented in this thesis. In the experiments described in this section, the cross validation function is used. It is mainly used in machine learning applied to estimate the ability of a model in unseen data [7]. It is a popular method because it is simple to understand and generally results in a less biased estimate than other methods, such as the simple division of training and testing.

First, new experiments are carried out with the Weigthed Voting algorithm, to determine the weights to assign to each classifier. To do this, a process is run testing different weights for each of the classifiers. The results were compared to choose the best model.

In Table 6.2 it can be seen that the best results, as in the previous stage, are given by assigning the weights as follows: 1 to Logistic Regression, 3 to Random Forest and 1 to Naive Bayes, when using Weighted Voting as the Ensemble Learnig algorithm. Therefore, this is the model chosen to perform the Phase 1 ensembling.

LR	RF	NB	Accuracy
1	1	2	60%
1	1	3	58%
1	2	1	99.5%
1	2	2	99.3%
1	3	1	99.98%
1	2	2	82%
2	1	1	91.3%
2	1	3	96.8%
2	2	2	90.48%
2	3	3	97.25%

Table 6. 2:Experiments using Logistic Regression (LR), Random Forest (RF) and Naive Bayes (NB) as base algorithms and applying Weighted Voting as Ensembling techniques. All possible combinations of weights are tested with values 1, 2 and 3. The best value of accuracy.

In the next two phases we will list the results as we did in this section (Phase 1).

6.3. Phase 2 Testing: Malicious or Normal Based on Flow from Origin to a Destination

Logistic Regression: **Accuracy 99,69%**

Random Forest: **Accuracy 98.3%**

Naive Bayes: **Accuracy 98.145%**

K-Nearest Neighbor: **Accuracy 99.23%**

Decision Tree: **Accuracy 97.125%**

The results above shows the precision obtained for the following automatic learning algorithms: Logistic Regression, Random Forest, Naive Bayes, K Nearest Neighbor and Decision Tree. These values are then compared with those resulting from applying different Ensemble Learning techniques to combine them.

Majority Voting	**99,584%**
Soft Voting	**97,129%**
Weighted Voting (LR=1, RF=3 y NB=1)	**98,97%**
Weighted Voting (LR=1, RF=3 y NB=2)	**99.215%**
Weighted Voting (LR=2, RF=3 y NB=1)	**99.97%**

Results of the Experiments using Logistic Regression, Random Forest and Naive Bayes as base algorithms and applying different variants of voting techniques as Ensemble Learning. The first is shown by the value of Accuracy when applying Majority Voting. The second sample shows the Accuracy value when applying Soft Voting. The following show the Accuracy value for Weighted Voting with different weights assigned to Logistic Regression, Random Forest, and Naive Bayes. These correspond to the weights for which the best Accuracy values were obtained

Weighted Voting (LR=1, KNN=2 y NB=1)	**91,784%**
Weighted Voting (LR=1, KNN=3 y NB=1)	**96,458%**
Weighted Voting (LR=2, KNN=3 y NB=1)	**98,945%**

Results of the experiments using Logistic Regression, K-Nearest Neighbor and Naive Bayes as base algorithms, and applying Weighted Voting as the Ensembling technique. In these experiments, the Random Forest algorithm was replaced by K Nearest Neighbor.

Weighted Voting (LR=1, DT=3 y NB=1) **98.361%**
Weighted Voting (LR=1, DT=3 y NB=2) **98.115%**
Weighted Voting (LR=2, DT=3 y NB=1) **99.321%**

Results of the experiments using Logistic Regression, Decision Tree and Naive Bayes as base algorithms and applying Weighted Voting as an Ensembling technique. In these experiments, the Random Forest algorithm was replaced by Decision Tree.

Applying Bagging results in minor improvements for Random Forest, KNN, and DT, and no improvement for Logistic Regression and Naive Bayes. The results are shown below in table 6.3.

Algorithm	With Bagging	Without Bagging
LR	96,015%	99,898%
RF	98,785%	98,215%
NB	95,547%	96,257%
KNN	98.964%	91,917%
DT	97.152%	94,158%

Table 6. 3: Experiments using Logistic Regression, Random Forest, Naive Bayes, K Nearest Neighbor and Decision Tree as base algorithms and applying Bagging as Ensembling technique.

6.4. Phase 2 Testing

LR	RF	NB	Accuracy
1	1	2	99.8%
1	1	3	45.4%
1	2	1	65.21%
1	2	2	96.31%
1	3	1	92.15%
1	2	2	78.54%
2	1	1	78.23%

2	1	3	98.78%
2	2	2	96.89%
2	3	3	98.25%

Table 6. 4: Experiments using Logistic Regression (LR), Random Forest (RF) and Naive Bayes (NB) as base algorithms and applying Weighted Voting as Ensembling techniques. All possible combinations of weights are tested with values 1, 1 and 2. The best value of accuracy.

6.5. Phase 2 Testing: Malicious or Normal Based source IP is Infected

Logistic Regression: **Accuracy 91,21%**

Random Forest: **Accuracy 90.5%**

Naive Bayes: **Accuracy 93.12%**

K-Nearest Neighbor: **Accuracy 96.189%**

Decision Tree: **Accuracy 97.25%**

The results above shows the precision obtained for the following automatic learning algorithms: Logistic Regression, Random Forest, Naive Bayes, K Nearest Neighbor and Decision Tree. These values are then compared with those resulting from applying different Ensemble Learning techniques to combine them.

Majority Voting **98.25%**
Soft Voting **98.712%**
Weighted Voting (LR=1, RF=3 y NB=1) **98.962%**
Weighted Voting (LR=1, RF=3 y NB=2) **96.897%**
Weighted Voting (LR=2, RF=3 y NB=1) **94.551%**

Results of the Experiments using Logistic Regression, Random Forest and Naive Bayes as base algorithms and applying different variants of voting techniques as Ensemble Learning. The first is shown by the value of Accuracy when applying Majority Voting. The second sample shows the Accuracy value when applying Soft Voting. The following show the Accuracy value for Weighted Voting with different

weights assigned to Logistic Regression, Random Forest, and Naive Bayes. These correspond to the weights for which the best Accuracy values were obtained

Weighted Voting (LR=1, KNN=2 y NB=1) 96.125%
Weighted Voting (LR=1, KNN=3 y NB=1) 98.025%
Weighted Voting (LR=2, KNN=3 y NB=1) 98.969%

Results of the experiments using Logistic Regression, K-Nearest Neighbor and Naive Bayes as base algorithms, and applying Weighted Voting as the Ensembling technique. In these experiments, the Random Forest algorithm was replaced by K Nearest Neighbor.

Weighted Voting (LR=1, DT=3 y NB=1) 98.628%
Weighted Voting (LR=1, DT=3 y NB=2) 97.854%
Weighted Voting (LR=2, DT=3 y NB=1) 98.785%

Results of the experiments using Logistic Regression, Decision Tree and Naive Bayes as base algorithms and applying Weighted Voting as an Ensembling technique. In these experiments, the Random Forest algorithm was replaced by Decision Tree.

Applying Bagging results in minor improvements for Random Forest, KNN, and DT, and no improvement for Logistic Regression and Naive Bayes. The results are shown below in table 6.5.

Algorithm	With Bagging	Without Bagging
LR	96.115%	99.785%
RF	98.545%	96.25%
NB	92.521%	98.45%
KNN	98.905%	97.12%
DT	98.21%	98.63%

Table 6. 5: Experiments using Logistic Regression, Random Forest, Naive Bayes, K Nearest Neighbor and Decision Tree as base algorithms and applying Bagging as Ensembling technique.

6.6. Phase 3 testing

LR	RF	NB	Accuracy
1	1	2	90.2%
1	1	3	69.884%
1	2	1	96.35%
1	2	2	87.25%
1	3	1	77.95%
1	2	2	99.1%
2	1	1	96.2%
2	1	3	98.21%
2	2	2	96.47%
2	3	3	99.58%

Table 6. 6: Experiments using Logistic Regression (LR), Random Forest (RF) and Naive Bayes (NB) as base algorithms and applying Weighted Voting as Ensembling techniques. All possible combinations of weights are tested with values 2, 3 and 3. The best value of accuracy.

Chapter Seven: Conclusion and Future Works

At the end of the thesis, the objectives set at the beginning have been met, from the creation of a methodology to detect infected hosts on the network, which Ensemble Learning applies in the different phases of the process. And its implementation in a module integrated into Slips that provides the tool with the benefits of Ensembling.

The method defined in this work was validated through experiments using datasets with real traffic data, provided by Stratosphere [48] and available to the community. This allows you to adjust the process taking into account flows that represent the real behavior of a set of hosts and learn from it.

This proposal for the implementation of ensembling in phases allows us to solve an aspect of the problem in each of them, and in turn take the predictions of the previous phase, which are combined with the analysis of the phase itself to achieve better results.

The training that is carried out in each phase to choose the best model allows the model to be adjusted to detect new attacks. This puts it at an advantage over static rule-based or signature-based tools.

Said mechanism is novel when combining: the information provided by the datasets generated from the traffic of infected machines and non-infected machines (Phase 1), the information provided by the analysis of the TCP connections established, TCP not established, UDP established and UDP not established with each destination with which a host communicates (Phase 2), and the information that the different IT sources provide in relation to the destinations with which the host communicates. a host connects (Phase 3).

This combination, which is done to make the final decision of whether a host is infected or not, is to apply Ensembling itself. It is decided taking into account the results of the 3 phases.

There are advantages of each phase. In Phase 1, when using ensembling to classify the owes, a decision is implemented that combines the findings of the base classifiers

implemented in Slips. In particular the selected technique of voting with weight, choose for this case the decision of the base algorithm that best classificatiom.

In Phase 2, the ensembling is done by grouping flows by destination, and in each of these groups, by protocol and state. This allows malicious connections to be detected based on the characteristics of the connections between the source and the destination. In turn, that the decision is by destination, leads to adding in Phase 3 the information from Threat Intelligence for each of them.

Applying ensembling in Phase 3 with IT information enriches the knowledge of the origin host's behavior, as reported by the Threat Intelligence feeds regarding the destinations with which it connects. By dividing the problem into phases, the model becomes adaptable because it is possible to improve the technique of ensembling in one of the phases without affecting the others. The

The training of each phase is independent, with which the model of a phase can be changed to improve detection without affecting the definition of the other phases, although it is improving, improving its results as a consequence.

Regarding my professional development, this work motivates me to continue this line of work, related to data intelligence and cybersecurity. Within this framework, a set of possible future lines of research and development are proposed:

- Include other sources of threat intelligence in Phase 3 of the methodology, in order to improve the detection of infected hosts.
- Integrate Slips and the ensembling module as feed, based on the proposal made in this thesis.
- Analyze the effectiveness of the teaching methodology proposed as a line of research to train human resources.
- Analyze the applicability of the proposed mechanism for IoT trafficking, in order to detect infected devices.
- Generate datasets from information provided by different network monitoring tools, which can be used to apply machine learning techniques and ensemble learning in particular, in order to detect infected hosts.

- Study the applicability of ensembling learning to detect different network security attacks.

References

[1] Agarwal., R. (2019). "The 5 classification evaluation metrics every data scientist must know. and when exactly to use them,". IEEE, 1(52), pp 150-163.

[2] Babayeva., K. (2020). "Introducing kalipso: the new interactive GUI of the stratosphere linux IPS," Stratosphere Technological Research, 1(10), pp 78-95.

[3] Bonoro., G. (2018). "Machine Learning Algorithms," Packeter Publishing Agency, 20(95), pp 100-695.

[4] Bowles, B. and Neiyr., K. (2015). "Machine Learning in Python: Essential Techniques for Predictive Analytics,". wiley.

[5] Groeey., I. (2019). "Threat intelligence (TI): What it is, and how to use it effectively," Springer.

[6] Brolee., J. (2015). "Machine Learning Mastery with Python," Google Books.

[7] Brolee., J. (2019). "an introduction to scikit-learnk-fold cross-validation.

[8] Ngen., C. (2018). "incident report system,".

[9] Ramach., A. et al. (2006). "Filtering spam with behavioral blacklisting,". -ACM conference on Computer and communications security, pp 25-34.

[10] Neale., C. "Cross-validation," towards data science hub.

[11] Vasilomanolakis., E. (2015). "Taxonomy and survey of collaborative intrusion detection," ACM Computing Surveys, 15(1).

[12] Bahri, E. et al. (2011). "Approach-based ensemble methods for better and faster intrusion detection," Intelligence in Security for Information and Security, pp 52-59.

[13] Bahri, E. et al. (2017). "A survey of intrusion detection systems based on ensemble and hybrid classifiers," Computers Security Journal IDRS.

[14] Grion, H. et al. (2020). "Quality evaluation of cyber threat intelligence feeds,". International Conference on Applied Cryptography and Network Security (ACNS).

[15] Quittek., J. et al. (2008). "Requirements for ip flow information export (ipx)," rfc 3917 (informational).

[16] Buitinck., L. et al. (2013). "Design for machine learning software: experiences from the scikit-learn project,". ECML Workshop: Languages for Data Mining and Machine Learning, pp 65-98.

[17] Didaci., L. et al. (2000). "Ensemble learning for intrusion detection in computer networks,".

[18] Aljarrah, O. et al. (2015). "Data randomization and cluster-based partitioning for botnet intrusion detection," IEEE Trans CyberNET, 2(22), pp 10-16.

[19] Venosa., P. et al. (2019). "Ensembling to improve infected hosts detection," CACIC 2019, pp 1251-1260.

[20] Venosa., P. et al. (2020). "A better-infected hosts detection combining ensemble learning and threat intelligence," Springer, pp 229-236.

[21] Abu., S. et al. (2018). "Cyber threat intelligence: issue and challenges,". Journal of Electrical Engineering and Computer Science.

[22] Ganapathy., S. et al. (2013). "Intelligent feature selection and classification techniques for intrusion detection in networks: a survey," EURASIP Journal on Wireless Communications and Networking, pp 271-282.

[23] Chih-Fong., T. et al. (2009). "Intrusion detection by machine learning: A review," Expert Systems with Applications.

[24] Sabatino., G. (2016). "Review ensemble-based collaborative and distributed intrusion detection systems: A survey," Journal of Network and Computer Applications. Elsevier Journal, pp 61-72.

[25] Shadow Server Foundation.

[26] Heras., J. (2020). "Ensembles: voting, bagging, boosting, stacking,"

[27] Hollmannov., D. (2020). "Writing slips module. Stratosphere Research Blog,".

[28] Humphries., S. (2020). "Threat intelligence feeds: Keeping ahead of the attacker,".

[29] Ponemon Institute. (2020). "The second annual study on exchanging cyber threat intelligence: There has to be a better way,".

[30] Stratosphere Lab. "Malware Capture Facility Project,".

[31] Stratosphere Lab. "Stratosphere IPS,".

[32] Stratosphere Lab. (2019). "Stratosphere Linux IPS (Slips) version 0.6.8,".

[33] Stratosphere Lab. (2019). "Stratosphere research laboratory,".

[34] Listvan., R. (2020). "Introducing flow formats and their differences,".

[35] Noh., L. (2017). "Cluster ensemble with a link-based approach for

botnet detection," Journal of Network and Systems Management, pp 550-562.

[36] Earle., S. (2017). "Multi-perspective machine learning

a classifier ensemble method for intrusion detection,".

[37] Earle., S. (2017). "Multi-perspective machine learning (mpml) a machine learning model for multi-faceted learning problems. pp 363-368.

[38] Okhan., D. (2018). "Modi

ed stacking ensemble approach to detect

network intrusion," Turkish Journal of Electrical Engineering & Computer Sciences, pp 418-433.

[39] online confusion matrix project. (2021). "confusion matrix CONCEPTS,".

[40] opensource.com project. (2020). "What is docker?,".

[41] Spamhaus Organization. (2020). "The Spamhaus Project,".

[42] Pokorny., Z. (2019). "The definition of is threat intelligence,".

[43] Zeek Project. Online Source, Last Access (Augest 2021).

[44] Zeek Project. (2019). "Zeek: an open-source network security monitoring tool,".

[45] Sirture., F. (2020). "Illustration of a boosting method for ensemble learning," IEEE Online conference, 14(20), pp 10-18.

[46] Sirture., F. (2020). "Illustration of a bootstrap aggregating

(bagging) method for ensemble learning," IEEE Online conference, 14(20), pp 19-29.

[47] SOURCE FORGE. (2020).

[48] Stratosphere. (2019). "Stratosphere laboratory datasets,".

[49] Stunga., S. (2016). "stacking methods in machine learning models," ACM, technology, and research. pp 89-102.

[50] Rais., W. (2018). "A survey on technical threat intelligence in the age of sophisticated cyber attacks," Computers security, 72(212).

[51] Casas., J. (2017). "Ensemble-learning approaches for network security and anomaly detection," ACM In: Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks, 1(1), pp 1-16.

[52] VirusTotal. Reports. (2019).

[53] VirusTotal. Virustotal home page.

[54] Weisberg., J. (2019). "Argus- the all seeing,".

[55] Ma., C. (2012). "Ensemble Machine Learning: Methods and Applications," Springer.

[56] Zhou., Z. (2012). "Ensemble Methods: Foundations and Algorithms," Google Books.

Appendix A: Python Code Functions

```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Input, ZeroPadding2D, BatchNorm
alization, Activation, MaxPooling2D, Flatten, Dense
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.utils import shuffle
import cv2
import imutils
import numpy as np
import matplotlib.pyplot as plt
import time
from os import listdir

%matplotlib inline


def evaluate_algorithm(dataset, algorithm, n_folds, *args):
	folds = cross_validation_split(dataset, n_folds)
	scores = list()
	for fold in folds:
		train_set = list(folds)
		train_set.remove(fold)
		train_set = sum(train_set, [])
		test_set = list()
		for row in fold:
			row_copy = list(row)
			test_set.append(row_copy)
			row_copy[-1] = None
		predicted = algorithm(train_set, test_set, *args)
```

```
            actual = [row[-1] for row in fold]
            accuracy = accuracy_metric(actual, predicted)
            scores.append(accuracy)
      return scores

def predict(summaries, row):
      probabilities = calculate_class_probabilities(summaries, row)
      best_label, best_prob = None, -1
      for class_value, probability in probabilities.items():
            if best_label is None or probability > best_prob:
                  best_prob = probability
                  best_label = class_value
      return best_label

def calculate_class_probabilities(summaries, row):
      total_rows = sum([summaries[label][0][2] for label in summaries])
      probabilities = dict()
      for class_value, class_summaries in summaries.items():
            probabilities[class_value] =
summaries[class_value][0][2]/float(total_rows)
            for i in range(len(class_summaries)):
                  mean, stdev, _ = class_summaries[i]
                  probabilities[class_value] *=
calculate_probability(row[i], mean, stdev)
      return probabilities

def predict(summaries, row):
      probabilities = calculate_class_probabilities(summaries, row)
      best_label, best_prob = None, -1
      for class_value, probability in probabilities.items():
            if best_label is None or probability > best_prob:
                  best_prob = probability
                  best_label = class_value
      return best_label

def data_percentage(y):

    m=len(y)
    n_positive = np.sum(y)
    n_negative = m - n_positive

    pos_prec = (n_positive* 100.0)/ m
    neg_prec = (n_negative* 100.0)/ m

    print(f"Number of examples: {m}")
    print(f"Percentage of positive examples: {pos_prec}%, number of pos exa
mples: {n_positive}")
    print(f"Percentage of negative examples: {neg_prec}%, number of neg exa
mples: {n_negative}")
```

Appendix B: Train History for Naïve Bayes Algorithm (as a sample)

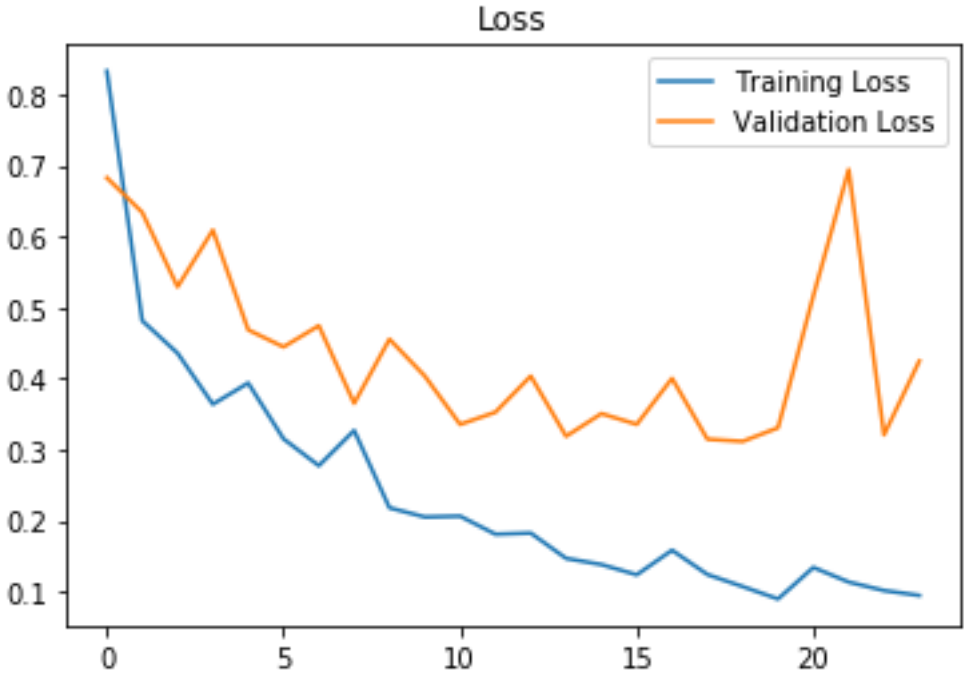
```
Training Data:
Number of examples: 1445
Percentage of positive examples: 52.8719723183391%, number of pos examples:
764
Percentage of negative examples: 47.1280276816609%, number of neg examples:
681
Validation Data:
Number of examples: 310
Percentage of positive examples: 54.83870967741935%, number of pos
examples: 170
Percentage of negative examples: 45.16129032258065%, number of neg
examples: 140
Testing Data:
Number of examples: 310
Percentage of positive examples: 48.70967741935484%, number of pos
examples: 151
Percentage of negative examples: 51.29032258064516%, number of neg
examples: 159

number of training examples = 1445
number of development examples = 310
number of test examples = 310
X_train shape: (1445, 240, 240, 3)
Y_train shape: (1445, 1)
X_val (dev) shape: (310, 240, 240, 3)
Y_val (dev) shape: (310, 1)
X_test shape: (310, 240, 240, 3)
Y_test shape: (310, 1)
```

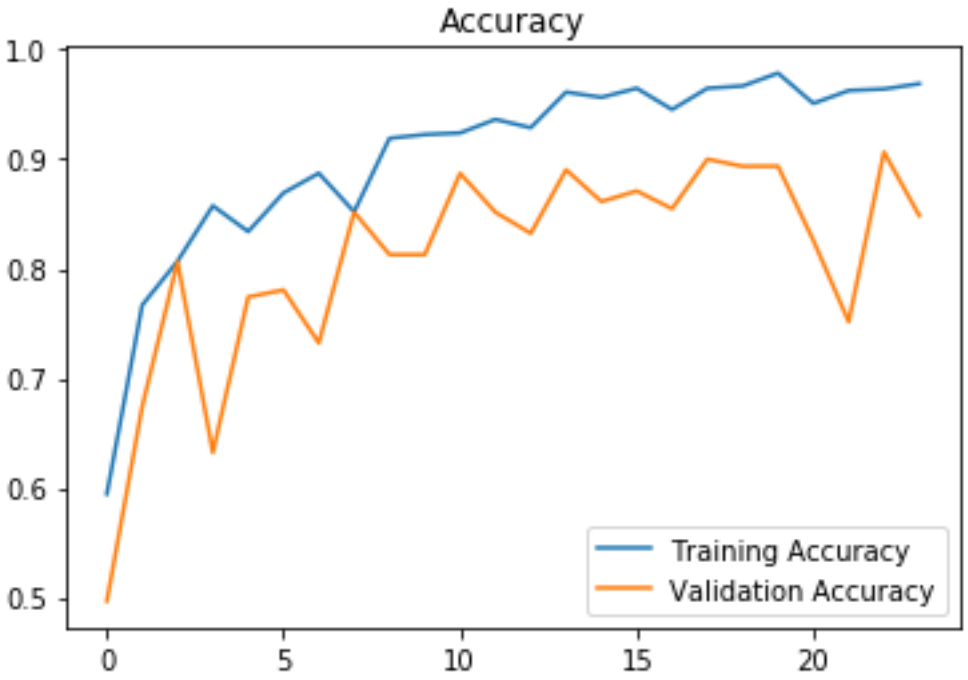
Validation set	Test set	
Accuracy	91%	89%
F1 score	0.91	0.88

```
Train on 1445 samples, validate on 310 samples
Epoch 1/10
1445/1445 [==============================] - 434s 300ms/step - loss: 0.8331
- acc: 0.5945 - val_loss: 0.6829 - val_acc: 0.4968
Epoch 2/10
1445/1445 [==============================] - 463s 320ms/step - loss: 0.4817
- acc: 0.7668 - val_loss: 0.6342 - val_acc: 0.6742
Epoch 3/10
1445/1445 [==============================] - 471s 326ms/step - loss: 0.4361
- acc: 0.8069 - val_loss: 0.5294 - val_acc: 0.8065
Epoch 4/10
1445/1445 [==============================] - 465s 322ms/step - loss: 0.3641
- acc: 0.8574 - val_loss: 0.6092 - val_acc: 0.6323
Epoch 5/10
1445/1445 [==============================] - 457s 316ms/step - loss: 0.3940
- acc: 0.8339 - val_loss: 0.4689 - val_acc: 0.7742
Epoch 6/10
```

```
1445/1445 [==============================] - 452s 313ms/step - loss: 0.3154
- acc: 0.8692 - val_loss: 0.4448 - val_acc: 0.7806
Epoch 7/10
1445/1445 [==============================] - 465s 322ms/step - loss: 0.2776
- acc: 0.8872 - val_loss: 0.4747 - val_acc: 0.7323
Epoch 8/10
1445/1445 [==============================] - 439s 304ms/step - loss: 0.3271
- acc: 0.8519 - val_loss: 0.3655 - val_acc: 0.8516
Epoch 9/10
1445/1445 [==============================] - 435s 301ms/step - loss: 0.2182
- acc: 0.9190 - val_loss: 0.4557 - val_acc: 0.8129
Epoch 10/10
1445/1445 [==============================] - 438s 303ms/step - loss: 0.2054
- acc: 0.9225 - val_loss: 0.4038 - val_acc: 0.8129
Elapsed time: 1:15:23.8

Train on 1445 samples, validate on 310 samples
Epoch 1/3
1445/1445 [==============================] - 431s 299ms/step - loss: 0.2065
- acc: 0.9239 - val_loss: 0.3357 - val_acc: 0.8871
Epoch 2/3
1445/1445 [==============================] - 432s 299ms/step - loss: 0.1811
- acc: 0.9363 - val_loss: 0.3529 - val_acc: 0.8516
Epoch 3/3
1445/1445 [==============================] - 425s 294ms/step - loss: 0.1827
- acc: 0.9287 - val_loss: 0.4038 - val_acc: 0.8323
Elapsed time: 0:21:29.4

Train on 1445 samples, validate on 310 samples
Epoch 1/3
1445/1445 [==============================] - 438s 303ms/step - loss: 0.1471
- acc: 0.9612 - val_loss: 0.3190 - val_acc: 0.8903
Epoch 2/3
1445/1445 [==============================] - 432s 299ms/step - loss: 0.1384
- acc: 0.9564 - val_loss: 0.3509 - val_acc: 0.8613
Epoch 3/3
1445/1445 [==============================] - 429s 297ms/step - loss: 0.1240
- acc: 0.9647 - val_loss: 0.3358 - val_acc: 0.8710
Elapsed time: 0:21:38.5

Train on 1445 samples, validate on 310 samples
Epoch 1/3
1445/1445 [==============================] - 536s 371ms/step - loss: 0.1586
- acc: 0.9453 - val_loss: 0.4005 - val_acc: 0.8548
Epoch 2/3
1445/1445 [==============================] - 427s 296ms/step - loss: 0.1244
- acc: 0.9647 - val_loss: 0.3149 - val_acc: 0.9000
Epoch 3/3
1445/1445 [==============================] - 429s 297ms/step - loss: 0.1074
- acc: 0.9668 - val_loss: 0.3118 - val_acc: 0.8935
Elapsed time: 0:23:11.9

Train on 1445 samples, validate on 310 samples
Epoch 1/5
1445/1445 [==============================] - 427s 296ms/step - loss: 0.0899
- acc: 0.9785 - val_loss: 0.3310 - val_acc: 0.8935
Epoch 2/5
1445/1445 [==============================] - 426s 295ms/step - loss: 0.1343
- acc: 0.9509 - val_loss: 0.5169 - val_acc: 0.8258
```

```
Epoch 3/5
1445/1445 [==============================] - 425s 294ms/step - loss: 0.1137
- acc: 0.9626 - val_loss: 0.6945 - val_acc: 0.7516
Epoch 4/5
1445/1445 [==============================] - 430s 298ms/step - loss: 0.1018
- acc: 0.9640 - val_loss: 0.3210 - val_acc: 0.9065
Epoch 5/5
1445/1445 [==============================] - 434s 300ms/step - loss: 0.0949
- acc: 0.9689 - val_loss: 0.4250 - val_acc: 0.8484
Elapsed time: 0:35:41.9
```





```
Test Loss = 0.33390871454631127
Test Accuracy = 0.8870967741935484
```