ALMA MATER STUDIORUM · UNIVERSITY OF BOLOGNA

SCHOOL OF SCIENCE

Second degree in computer science

Applying Machine Learning to Cyber Security

Relator:
Prof.
Paolo Ciancarini

Correlators:
Prof.
Valentina Presutti
Mehwish Alam

Presented by:
Carlo Stomeo

Session II
Academic Year 2017-2018

"Be able to fly, learning to fall"

Salmo

Contents

List of Figures

List of Tables

Italian Introduction - Introduzione

Parallelamente alla crescita esponenziale del Web e dei servizi ad esso associati, crescono anche gli attacchi informatici. Per questo motivo l'utilizzo di misure di sicurezza non è mai stato così importante. Al giorno d'oggi le tecniche di Intrusion Detection (IDS) e di Vulnerability Detection sono due delle misure di sicurezza più utilizzate. Gli IDS hanno lo scopo di analizzare un sistema o una rete, alla ricerca di minacce alla sicurezza, come attività sospette e accessi non autorizzati. Mentre la Vulnerability Detection consiste nell'analisi di un sistema, alla ricerca di vulnerabilità, punti deboli, che possono essere sfruttati da un utente malintenzionato. Sono state proposte molte tecniche per raggiungere questi obiettivi e ora tra queste stanno emergendo anche Machine Learning (ML) e Data Mining (DM). Maggiori dettagli sugli IDS (1.1) e sulla Vulnerability Detection (1.2) sono riportati nel capitolo seguente.

Questo lavoro si pone due goal principali:

- Analizzare lo stato dell'arte su techiche di ML e DM applicate in questi campi della cyber security.
  - Fare una survey.
  - Valutare la riproducibilità dei metodi analizzati.
  - Comparare alcuni dei metodi analizzati nel medesimo scenario.
- Proporre una soluzione ad un caso d'uso reale.

Il primo passo non consiste semplicemente nella stesura di un Survey. Questo perché la valutazione e la vera comprensione dello stato dell'arte sono molto importanti, ma per ottenere realmente un vantaggio e una comprensione di ciò che è già stato fatto, dobbiamo riprodurre e confrontare i metodi proposti. Questo porta ad un altro importante aspetto nell'ambito del Machine Leaning e più in generale in informatica: l'adesione ai principi FAIR [9]. Questi principi sono stati proposti al fine di fornire una serie di linee guida per rendere i dati reperibili, accessibili, interoperabili e riutilizzabili. Questo non si applica solo ai dati, ma anche al codice, che dovrebbe essere open source e quindi riproducibile, almeno nell'area di ricerca, consentendo così una crescita continua nello sviluppo di nuove soluzioni. (Una discussione riguardo l'adesione dei metodi proposti nello stato dell'arte a queste linee guida può essere trovata in sezione 3.8.)

Gli articoli presi in esame sono stati scelti per il loro impatto, considerando il conteggio delle citazioni e il livello della conferenza o del giornale in cui sono stati pubblicati. Inoltre, anche la data di pubblicazione è stata considerata nella fase di scelta.
Per ragioni di tempo e coerenza con il caso d'uso reale analizzato, anche se il lo stato dell'arte è stato analizzato sia per gli IDS che per la Vulnerability Detection, il confronto sperimentale è stato fatto solo per i primi, selezionando i tre approcci che sono stati meglio documentati dagli autori. Il confronto (capitolo 5) si è svolto utilizzando la stessa macchina e gli stessi dati, al fine di confrontare le prestazioni dei metodi proposti nello stesso scenario. Da questa prima analisi le tecniche di outlier detection sono risultate le più efficiaci nell'individuare attachi noti e non. Infine, le conoscenze acquisite da questo primo studio sono state utilizzate per trovare una soluzione a uno scenario reale (capitolo 6). La scopo di quest'ultima fase è stato quello di creare un IDS utilizzando solo i log di un server Apache in modo completamente non supervisionato. La soluzione proposta consiste in due parti: *(i)* la pre-elaborazione dei dati e *(ii)* la proposta di un modello non supervisionato che utilizza tecniche di anomaly-outlier detection.

Introduction

In parallel to the exponential growth of the web and web based services also cyber attacks are growing. For this reason the usage of security measures has never been so important. Intrusion Detection Systems (IDS) and Vulnerability detection techniques are two of the most used security measures nowadays. IDSs have the purpose of analyzing a system or a network looking for security threats, like suspicious activity and unauthorized access. While Vulnerability Detection consists in the analysis of a system, looking for vulnerabilities, weaknesses that can be exploited by an attacker. Many techniques to achieve these goals have been proposed and now also Machine Learning (ML) and Data Mining (DM) ones are emerging. For more details about IDS (1.1) and Vulnerability Detection (1.2) refer to the next chapter. This work has two main goals:

- Assess the state of the art about ML and DM techniques applied to these cyber security fields.
  - Make a Survey.
  - Consider the reproducibility of the proposed methods.
  - Compare some of the proposed methods with a common experimental setting.
- Propose a solution to a real world scenario.

As you can see, the first step does not simply consist in the making of a Survey, this because the Assessment of the state of the art is very important.

But to really gain advantage and comprehension of what has already been done we need to reproduce and compare the proposed methods. This leads to another important aspect in Machine Leaning and more in general in Computer Science: the adhesion to the FAIR principles [9]. This principles have been proposed in order to give a set of guidelines to make data findable, accessible, interoperable and reusable. This does not only apply to the data, but also to code, which should be open source and so reproducible, at least in the research area, thus allowing a continuous growth in the development of new solutions. (For a discussion about this aspects refer to section 3.8)
The papers taken under exam have been chosen for their impact, considering the citation count, and the rank of conference/journal in which they have been published. Moreover also the publication date has been considered as an important aspect.
For time reasons and coherence to the real world scenario, even if the Survey has been done for both IDSs and Vulnerability Detection, the experimental comparison has been done only for IDSs, selecting the three of them which have been better documented by the authors. The comparison in chapter 5 has been done with the same machine and the same data, in order to compare the performances of the proposed methods in the same scenario. Finally the knowledge acquired from this first study has been used to find a solution to a real world scenario (chapter 6). The task was to make an IDS using only web server logs in a complete unsupervised way. The proposed solution consists in two parts: *(i)* the data preprocessing and *(ii)* the proposal of an unsupervised model using outlier detection.

Chapter 1

Background: Vulnerability, Intrusion detection and machine learning

This chapter explains the basic concepts needed to understand this whole work. As first Intrusion Detection Systems and software Anomaly Detection will be introduced. Then an overview about Machine Learning will be given.

1.1 Intrusion Detection Systems

An intrusion detection system (IDS) is a system that monitors network traffic for suspicious activity and unauthorized access. IDSs can be classified into three major classes: (i) Active and Passive IDS, (ii) Network Intrusion Detection Systems (NIDS) and (iii) Host-Intrusion Detection Systems (HIDS).

Active and Passive IDS: Active IDS are also known as Intrusion and Prevention Detection System (IPDS). They automatically block the suspected intrusions without an intervention of an operator. On the other hand, passive IDS only monitors and analyses the traffic and alerts an operator in case of an attack.

Host-Based IDS: These systems are installed on individual devices which are connected to the network. They monitor the traffic of each of the devices and they are deemed better if the activity of a particular device is to be monitored.

Network-Based IDS: These kinds of systems usually monitor all the passing traffic at strategic points of the networks.

Moreover IDSs can be grouped in three other categories basing on the method used to detect the attacks.

1.1.1 Misuse Based Intrusion Detection

Also known as Signature Based IDS, this systems identify security issues from a set of known attacks and vulnerabilities. The idea is that every attack can be expressed through a fingerprint that it leaves behind him, and then this one can be used to identify new occurrences of the same attack. This method can be very powerful detecting known attacks but the set of known attack fingerprints needs to be continuously updated. Moreover even with an updated dataset, previously unseen attack won't be detected.

1.1.2 Anomaly Based and Hybrid Intrusion Detection

This approach is in some sense complementary to the previous one. Instead of detecting attacks from a set of known ones, it uses the pattern of normal system behavior and marks as attack (anomaly) everything that deviates from that behavior. The pro of this technique is that it is able to detect previously unseen attacks. However there are also negative aspects doing this, for example there will always be new legitimate activities that will be marked as attacks. So the negative aspect of this system is that it often leads to a high number of false alarms. Another pro is that the normal usage pattern will be customized for every system, increasing the difficulty for the attackers to find activities that can be carried out undetected. Hybrid techniques are a combination of misuse-based and anomaly-based ones. They have been proposed to reduce the number of false alarms, while maintaining the capability to detect new attacks.

1.2 Vulnerability detection

"In the context of software security, vulnerabilities are specific flaws or oversights in a piece of software that allow attackers to do something malicious: expose or alter sensitive information, disrupt or destroy a system, or take control of a computer system or program." Dowd et al. [33].

Basically a vulnerability can be seen as a particular bug which can be exploited by a malicious user to start an attack against the system.
Vulnerability detection consists in the problem of analyzing a software and detect vulnerabilities contained in it. As Jhala and Majumdar explain in [28] this kind of problem is undecidable, this means that it is not possible to write a program that finds all the vulnerabilities (*soundness*) and reports no false vulnerabilities (*completeness*).
Despite this nature of the problem, as explained in [22] and summarized in [2] different approaches have been proposed, trying to find an approximate solution. This proposed methods can be divided in the following three main families.

1.2.1 Static Analysis

A program is analyzed based only on his source code. This means that there is no need to execute it. The approach examines the program code, applying specific rules or algorithms (also known as inference), and derives a list of vulnerable code present in a program that might result in successful exploitations (Shahriar and Zulkernine [22]). The effectiveness of any static

analysis depends on how accurate an inference technique is in discovering potential vulnerable code, moreover,as it always happens, there is a trade-off between the accuracy of the detection and the false positives. This means that static analysis can be at his best *sound*, but false vulnerabilities will be (probably) reported.
Basing on the inference technique adopted by the algorithm this methods can be divided as follows:

- **Tainted data-flow based** techniques mark input variables as tainted and track their propagations. Warnings are generated if tainted inputs or values derived from them are used in sensitive operations.

- **String-pattern-matching based** techniques derive from simple string pattern matching methods. These techniques use a set of known function calls which can cause vulnerabilities, and identify some vulnerable code starting from them. The program will then be tokenized and evaluated in search of these patterns.

- **Constraint based** techniques define a series of constraints from a set of known vulnerabilities in such a way that the violation of one of these constraints imply the presence of the related vulnerability. The constraints are then propagated and updated traversing the program and constraint solvers are used to find input values that can violate the constraints.

- **Annotation based** techniques annotate the program in terms of desired pre and postconditions. After this an algorithm checks if data variables can be used safely based on the annotated conditions or not. When a precondition cannot be resolved from a previous statement's postcondition, then a warning message is generated.

1.2.2 Dynamic Analysis

A program is analyzed executing it with some specific input and observing his runtime behavior. This kind of analysis is strongly dependent from the input, for this reason a Dynamic approach can't be *sound*. In fact, in the most of the cases, it is not possible to test a program with all the possible inputs (because they can be infinite), so there will always be the possibility that some vulnerability remains undiscovered. By the other way this approaches can be *complete*, approving all secure programs without false alarms.

1.2.3 Hybrid Analysis

Combining the two previous techniques is possible to combine the pros of both of them. This does not mean that Hybrid methods are both *complete* and *sound*, because as we said this is not possible. In fact as Hybrid analysis gains the advantages from both Static and Dynamic Analysis it also suffers the cons from both these methods. One approach to Hybrid Analysis can use Static Analysis to identify the locations in the program that may contain some vulnerabilities and that need to be analyzed during program executions to verify their actual exploitations (by Dynamic Analysis). In this way the number of suspected vulnerabilities reported by a static analysis can be reduced. A different approach can employ as first a Dynamic Analysis approach that leverages Static Analysis techniques to guide the test-case selection and analysis process.

1.3 Cyber Attacks

In parallel to the IT development a huge range of attacks showed up. The most common are briefly introduced in this section.

Brute Force Attack: The most basic kind of attack. It simply consist in a complete search over the credentials space, trying to discover the password or other informations.

Denial of Service (DoS): This attack has the purpose to exhaust the system's capabilities, causing the interruption of the supplied services. With DDoS a Distributed variant is referred. In this kind of attacks a huge amount of hosts (generally controlled by some malware) is used to generate thousands of request to a single target (typically a web server).

Code Injection: This attacks consists in the injection of some malign code in a web application with the aim to steal access credentials or to the impersonification of an already authenticated user.

Buffer Overflow: These attacks are characterized by the overwriting of memory fragments of the process. This kind of vulnerabilities can lead to DoS attacks or Code Injections.

Rootkit: a malign software with the aim to gain root access to a system. Sometimes it can also lead to the remote access and control to the attacked system.

Cross-Frame Scripting (XFS): An attack that combines Javascript code and an Iframe to load a legitimate page with the aim to steal user informations. It is often used with phishing techniques.

Cross-Site Scripting (XSS): it enables attackers to inject client-side scripts into web pages viewed by normal users. XSS is often used to bypass access controls such as the same-origin policy.

Keylogging: A keylogger is a software or hardware tool capable to secretly sniff and register all the characters the user is pushing on his keyboard.

Man in the middle: This attack consists in the interception of a communication between two users, staying in the middle of them and behaving with both of them as the other legitimate end of the communication.

Phishing: This term indicates the try to gain the credential of a user to stole his identity. The most typical phishing attack is made using an e-mail with looks legitimate with brings the user to a malign web page.

1.4 Machine Learning

Machine learning is a field of Artificial Intelligence that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Similarly to what happens to humans the learning process starts from some example, some data to work on. Based on the structure of this data Machine Learning algorithms mostly fall under the following categories:

Supervised Algorithms

In this family of approaches the algorithm learns from past knowledge. This means that the training data already contains knowledge, and the algorithms needs to learn from it to predict future events. In this case we are talking about labeled data, this means that the data comes also with his explanation, for example imagine to a dataset containing pictures, labeled with 1 if they contain a cat or 0 if not. If the aim of the program is to say whenever a picture contains a cat or not this is an example of labeled learning. In this kind of algorithms the aim becomes to learn a model to identify new occurrences, this model will basically be a function that explains the learning data.

Unsupervised Algorithms

Here the data does not contains additional information about his meaning so the purpose becomes to extract some pattern from unlabeled data. Let's think to the previous example, if the dataset only contains picture with or without cats, but they are not labeled we can still try to learn some pattern from the data which will probably identify all the pictures containing cats.

Semi-supervised Algorithms

This family of algorithms falls in-between the previous ones. Semi-supervised methods use both labeled and unlabeled data. Typically a small quantity of labeled data is used to improve the accuracy of the model. This solutions are well used when acquiring or learning on labeled data is resource-expensive, while obtaining unlabeled data is not.

1.4.1 ML applications

Another categorization of machine learning algorithms can be done basing on the kind of output we are expecting to obtain from the learned system.

Classification

One of the aim of supervised machine learning algorithm is to learn a model from labeled data in order to gain the capability to assign labels to unlabeled data, this process is called Classification. A very clear example can be the mail filter which has to classify the incoming traffic between spam and non-spam mails.
The classification process is typically organized in three phases:

- **Training:** a model is constructed over a dataset of labeled data, called training dataset.

- **Validation:**As many classification approaches depends on some parameters which can modify their results, called meta-parameters, a phase to tune them is needed. The validation phase is made to tune this meta-parameters to optimize the classification performances. This phase works on a specific dataset, which has to be different from the training and the testing ones, to prevent overfitting[1].

- **Testing:** this last phase is made to give an evaluation in terms of accuracy, recall and so on, of a specific classification model. Similarly to the previous case the used dataset should be independent from the previous two.

A technique used to prevent overfitting is cross-validation. It is typically used when the training data is large. Cross validation is used to estimate how accurately a model performs. It consists in the splitting of the data in two complementary sub-sets, one used for training and the other one for validating or testing. To reduce the variability of this process it is usually repeated many times with different partitions. One example is 10-fold cross validation, where the data set is divided into 10 folds of the same size, 9 of them are used as training data, while the other one is used for testing. This process is repeated 10 times, using at each step a different partition for testing. In this way the model comes evaluated 10 times, and his final performance are computed as the average of these 10 evaluations.

In the following paragraphs the different classification types and the related evaluation approaches will be introduced.

Binary Classification is the easiest form of classification, it is characterized by the task to recognize the membership of an element between two classes. For this family of problems the metrics are computed from the confusion matrix (table 1.1).
True Positive (TP) means that an element has been recognized as istace of the class under exam and it really is, and in the same way a True Negative (TN) means that the membership of the element to the class was correctly

[1]Overfitting is defined as the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably

evaluated negative. False Positive (FP) and False Negative (FN) are respectively indicating that the classification was wrongly positive and wrongly negative.
The deriving metrics are the following:

- **Accuracy** gives a measure of the percentage of instances that are classified correctly.

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

- **Precision** is the percentage of elements classified correctly over the total number of elements classified as belonging to the target class.

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

- **Recall** (or detection rate in the following chapters) is the ratio of items correctly classified as belonging to the target class over all the items that actually belong to that class.

$$Recall = \frac{TP}{TP + FN}$$

- **False Alarm Rate (FAR)** or false positive rate is the ratio of items incorrectly classified as member of the target class over the total number of items not belonging to it.

$$FAR = \frac{FP}{TN + FP}$$

In figure 1.1 you can find an image intuitively showing precision and recall metrics[2].

[2]The figure was taken from: `https://en.wikipedia.org/wiki/Precision_and_recall`

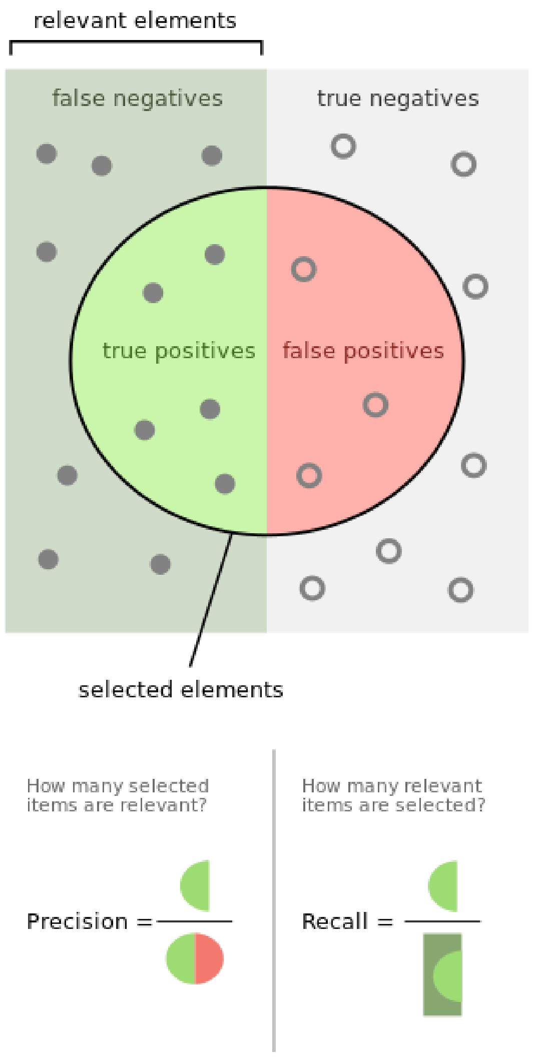


Figure 1.1: Precision and Recall metrics.

	Actual Positive	Actual Negative
Predicted Positive	TP	FP
Predicted Negative	FN	TN

Table 1.1: Confusion Matrix

Multi-class Classification consists in the problem to recognize the membership of an element between more than two classes. This can be achieved by combining multiple binary classifiers. There are two strategies to accomplish this: *one-vs-rest* (also known as *one-vs-all*) and *one-vs-one* (or *all-vs-all*). As the names suggest, the first strategy consists of training one classifier per class: the class being considered is the positive class, while the other classes are the negative class. The classification given to an element to evaluate is the class associated with the model that classifies that element with higher confidence. While in the second strategy, *one-vs-one* if n is the number of classes then *n*(n-1)/2* classifiers are trained, one for each combination of classes. During testing, the class that receives the highest number of votes is selected.
In this kind of classification the performance measures are:

- **Average Accuracy**

$$Accuracy_{average} = \frac{\sum_{i=1}^{l} \frac{TP_i+TN_i}{TP_i+TN_i+FP_i+FN_i}}{l}$$

- **Macro Precision**

$$Precision_{macro} = \frac{\sum_{i=1}^{l} \frac{TP_i}{TP_i+FP_i}}{l}$$

- **Macro Recall**

$$Recall_{macro} = \frac{\sum_{i=1}^{l} \frac{TP_i}{TP_i+FN_i}}{l}$$

- **Micro Precision**

$$Precision_{micro} = \frac{\sum_{i=1}^{l} TP_i}{\sum_{i=1}^{l} TP_i + FP_i}$$

- **Micro Recall**

$$Recall_{micro} = \frac{\sum_{i=1}^{l} TP_i}{\sum_{i=1}^{l} TP_i + FN_i}$$

- **Class False Positive Rate (Class FAR)** indicates the ratio of Exemplars from a given class incorrectly classified over all exemplars not from that given class.

The terms macro and micro are here indicating two different averaging strategies. The first one averages the respective measures calculated for each class, while the second one consists of calculating the sum of the numerators and the sum of the denominators of the respective measures calculated for each class, and then dividing the first sum by the second sum.

Multi Label Classification is a third case in which multiple labels may be assigned to each instance. Two are the approaches to classify multi-label data: *algorithm adaptation methods* and *problem transformation methods* [33]. The first approach accounts for ad-hoc versions of multi-class algorithms capable of generating multiple labels per instance. While the second one can be approached with different solutions. One of these consists of using *one-vs-rest* similar to the multi-class case. In this context, though, the output is the union of the "positive" decisions of the classifiers. Ad-hoc measures are required to evaluate the performance of a multi-label classifier [33]. If D is the dataset on which performance is to be evaluated, Y_i are the actual labels of the i^{th} instance of D and Z_i are the labels generated by the classifier on the same instance, accuracy, precision and recall respectively defined as:

$$Accuracy_{multi-label} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \bigcap Z_i|}{|Y_i \bigcup Z_i|}$$

$$Precision_{multi-label} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \bigcap Z_i|}{|Z_i|}$$

$$Recall_{multi-label} = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \bigcap Z_i|}{|Y_i|}$$

Clustering

Clustering consists in the problem to divide a set of inputs into groups. Unlike Classification the groups, and even their number, are not known a priori, so Clustering is an unsupervised problem. The groups (clusters) are made trying to maximize the similarity between members of the same cluster.

Similarly to classification, clustering is not a specific algorithm, but a task that can be achieved with different techniques, depending on the used model:

- **Connectivity models** (hierarchical clustering) group data by the distances between them.
- **Centroid models** (k-means) represents clusters by their mean vector.
- **Distribution models** (Expectation Maximization algorithm) assumes that the groups yield from a statistical distribution.
- **Density models** group the data points as dense and connected regions (Density-Based Spatial Clustering of Applications with Noise DBSCAN).
- **Graph models** (clique) defines each cluster as a set of connected nodes where each node has an edge to at least one other node in the set.

Regression

Regression analysis is a set of statistical processes for estimating the relationships among variables. In simple words the idea is that there are some independent variables, which, when taken together, produce a result - a dependent variable. The regression model is then used to predict the result of an unknown dependent variable, given the values of the independent ones. Similarly to classification Regression is a supervised problem, but it does not output a class, but a number. An example could be the problem of estimating the value of your house based on the number of bathrooms, bedrooms and the square footage, taking as learning dataset the houses for sale in your neighborhood.

1.4.2 ML approaches

To solve Machine Learning tasks a lot of possible approaches are available. In this work an approach is intended as a family of algorithms grouped for their similarities[3]. This categorization could not be complete, in sense that not all the possible ML algorithms fit in one of the following families, or some of them can fit into more than one, but it is useful to get an overall idea about the ML bases.

Regression Algorithms

As previously said, in this kind of supervised algorithms the purpose is to find a function capable to understand the relationship between a dependent variable and one or more independent variables (in the learning phase) and than to predict the value of an output variable (the dependent one), given some variable in input. While we speak about regression we can both refer to the class of problems and to the class of algorithms, the fact is that regression is a process, applied in algorithms to solve the class of problems.

Some of the most popular regression algorithms are:

- **Linear Regression:** it uses a linear approach to find the relationship between the independent variables and the dependent one.
- **Logistic Regression:** it is used to estimate the parameters of a logistic model. Differently from Linear regression where the output can assume continuous values, for logistic regression the problem is typically applied to a binary dependent variable, where the possible values are 0 and 1, representing outcomes such as win/fail, dead/alive or attack/normal.
- Stepwise Regression
- Ordinary Least Squares Regression (OLSR)
- Multivariate Adaptive Regression Splines (MARS)
- Locally Estimated Scatterplot Smoothing (LOESS)

[3]https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/ *accessed May 2018*

Instance-based Algorithms

This family of algorithms is characterized by the comparison of new problem instances with instances analyzed in the training phase and previously stored in memory. Typically a dataset of examples is made by the algorithm, and new data occurrences are compared with the one in the database using a similarity measure in order to find the best matching one.

Popular algorithms in this family are:

- **k-Nearest Neighbor (kNN):** this method solves the classification problem of an item basing on the classes of his k nearest neighbors in the multidimensional vector space defined from the items features.
- **Radial Basis Function Networks (RBF):** they are neural networks using RBF as activation function.
- Learning Vector Quantization (LVQ)
- Self-Organizing Map (SOM)
- Locally Weighted Learning (LWL)

Decision Tree Algorithms

This family of algorithms exploits a tree structure for his predicting process. The tree is used to go from some observation about an object (the branches of the tree) to conclusions about the item target value (represented in his leaves). Decision trees can be both used in classification problems (when the target variable can assume a discrete set of values) and regression problems (when the target variable can assume continuous values). More details about decision trees used in classification problems can be found at sec. 3.4. This family of algorithms is known to be fast and accurate, reason why it is a well used approach to many learning problems.

Some algorithms belonging to this family are:

- **Classification and Regression Tree (CART):** it consists in a decision tree implementation which can be used both for classification and regression problems.

- **Random Forest:** it consists in an ensemble method combing more decision trees. It is used both for classification and regression tasks and has been introduced to solve the decision trees' habit to over-fitting the training data.
- Iterative Dichotomiser 3 (ID3)
- Chi-squared Automatic Interaction Detection (CHAID)
- Conditional Decision Trees

Bayesian Algorithms

The methods in this family exploit the Bayes theorem to solve classification and regression problems. Briefly Bayesian inference derives the posterior probability as a consequence of two antecedents: a prior probability and a *likelihood function* derived from a statistical model for the observed data. The posterior probability is computed according to Bayes' theorem:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}$$

The most popular Bayesian algorithms are:

- **Naïve Bayes:** these methods are highly scalable. They require a number of parameters linear in the number of variables (features). Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time.
- **Gaussian Naïve Bayes:** a case of Naïve Bayes in which the continuous data are assumed to be distributed according to Gaussian distribution.
- **Multinomial Naïve Bayes:** a case of Naïve Bayes in which the data samples are assumed to be distributed according to Multinomial distribution.
- **Bayesian Network:** these methods use directed acyclic graph (DAG) to represents a set of variables and their conditional dependencies. The nodes of the DAG represent Bayesian variables, while the edges rep-

resent some probabilistic relationship. This means that if there is no path between two variables they are independent.

- Averaged One-Dependence Estimators (AODE)
- Bayesian Belief Network (BBN)

Clustering Algorithms

As for regression Clustering describes both a class of problem and a class of methods. For this reason Clustering has already been explained in subsection 1.4.1, so it won't be repeated.

Association Rule Learning Algorithms

Association rule learning methods extract rules that best explain observed relationships between variables in large databases. These rules can discover important associations in large multidimensional datasets using some measures of interest. A more detailed explanation of these algorithms can be found in sec. 3.2.

Two popular algorithms in this family are:

- **Apriori Algorithm:** it proceeds identifying frequent items in the data, extending them to larger datasets according to the fact that they must appear together a sufficient number of times in the data.
- **Eclat Algorithm:** Equivalence Class Transformation is faster than Apriori Algorithms using depth-first search algorithm based on set intersection.

Artificial Neural Networks Algorithms

Artificial Neural Networks (ANN) algorithms are inspired to the animal brain structure. The idea is to reproduce the behavior of biological neurons. An ANN is based on a collection of connected units or nodes called artificial neurons. Each neuron is a function that takes some input and give as a result an output that will be the input for the neurons in the next layer. Neurons are structured in layers and each neural network is composed by at least tree layers, one input layer, one (or more) hidden layer and one output layer. Basing on the number of hidden layers the ANN can be classified as normal or deep. In this second case the network is made by different neural layers. For more details about ANN look at sec. 3.1.

Some well known algorithms in this family are:

- **Convolutional Neural Network (CNN):** inspired to the animal visual cortex are typically used for image and video recognition.
- **Perceptron:** this algorithm is used to build linear binary classifiers.
- **Back-Propagation:** these techniques use back-propagation of the errors to improve the selection of the weights in the neural network.
- **Auto-Encoders:** this type of neural network is used to learn a representation for a set of data in an unsupervised way. It is typically used for dimensionality reduction tasks.

Dimensionality Reduction Algorithms

These algorithms act in a unsupervised way, trying to represent data in a simpler way. Basically they consist in the process of reducing the number of variables under exam. For this reason they are useful in the process of simplifying data for other supervised methods, like Classification and Regression.

Some known algorithms in this family are the following:

- **Principal Component Analysis (PCA):** this technique consists in a statistical procedure that uses an orthogonal transformation to reduce a set of observations of possibly correlated variables into a set of linearly uncorrelated variables.

- **Singular Value Decomposition (SVD):** SVD [4] is an algorithm that factors an m x n matrix (M) of real or complex values into three component matrices USV^*. U is an m x p matrix. S is a p x p diagonal matrix. V is an n x p matrix, with V^* being the transpose of V.

[4] https://blogs.oracle.com/r/using-svd-for-dimensionality-reduction *accessed July 2018*

- Principal Component Regression (PCR)
- Partial Least Squares Regression (PLSR)
- Sammon Mapping
- Multidimensional Scaling (MDS)
- Projection Pursuit
- Linear Discriminant Analysis (LDA)
- Mixture Discriminant Analysis (MDA)
- Quadratic Discriminant Analysis (QDA)
- Flexible Discriminant Analysis (FDA)

Ensemble Algorithms

Ensemble methods make use of multiple models trained independently and with different algorithms. The idea is to combine the single predictions in some way to obtain a better one. Some popular Ensemble Algorithms are:

- **Bootstrapped Aggregation (Bagging):** each model in the ensemble is trained on a random sub-set of the original training set. Moreover each model vote with the same weight. This kind of technique is used in random forest classifier.
- **Boosting:** the ensemble is built by incrementally training each new model focusing on the training instances that have been misclassified by the previous models. This methods are usually more accurate than Bagging, but they also tend more to over-fitting.
- **AdaBoost:** A famous implementation of boosting methods.
- Stacked Generalization (blending)
- Gradient Boosting Machines (GBM)
- Gradient Boosted Regression Trees (GBRT)
- Random Forest

Chapter 2

Data sources for intrusion detection

Similarly to what has been done in [6] in this chapter some available data-sources designed for evaluating IDS will be described, including both synthetic as well as real data. Moreover some simulator will be introduced. These last one can be used to generate attacks for testing the algorithms and for analysis purposes.

2.1 Packet Capture

The network traffic running on the different protocols like TCP, UDP, ICMP and so on can be analyzed and recorded by a specific API called pcap[1]. The Unix and Windows implementation of pcap are libpcacp[2] and WinPcap[3] which provide the packet-capture and filtering engines of many open source and commercial network tools, including protocol analyzers (packet sniffers), network monitors, network intrusion detection systems, traffic-generators and network-testers. One of the most important feature of this API is to provide the possibility of saving network logs on files which in turn allows a huge

[1] https://www.tcpdump.org/manpages/pcap.3pcap.html *accessed May 2018*
[2] http://www.tcpdump.org/ *accessed May 2018*
[3] https://www.winpcap.org/ *accessed May 2018*

quantity of network data to be further analyzed by the algorithms.

2.2 NetFlow Data

NetFlow is a feature introduced in Cisco[4] routers which collects IP network traffic as it enters or exits an interface. The network flow is defined as an unidirectional sequence of packets that share the exact same seven packet attributes: router/switch interface, source IP address, destination IP address, IP protocol, source port, destination port, and IP type of service. The Netflow architecture consists of three main components:

- **Flow exporter** aggregates packets into flows and exports flow records towards one or more flow collectors.
- **Flow collector** is responsible for reception, storage and preprocessing of flow data received from a flow exporter.
- **Analysis application** analyzes received flow data in the context of intrusion detection or traffic profiling, for example.

2.3 Public Datasets

The Cyber Systems and Technology Group of MIT Lincoln Laboratory, has collected and distributed the first standard corpora for evaluation of computer network intrusion detection systems. They provided two datasets as result of this evaluation: the Defense Advanced Research Projects Agency (DARPA) 1998 and DARPA 1999 data sets [40] [41]. These datasets contain extensive examples of attacks and background traffic in which the TCP dumps and logs were combined into one stream with many columns. The training dataset is labeled and the attacks fall into five main classes namely, Probe, Denial of Service(DoS), Remote to Local(R2L), User to Remote(U2R)

[4]https://www.cisco.com/ *accessed May 2018*

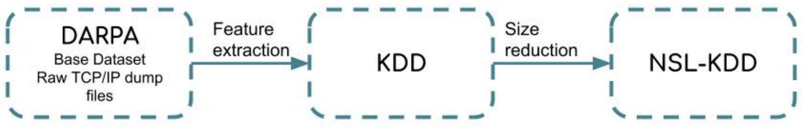


Figure 2.1: The relation between DARPA, KDD-99 and NSL-KDD datasets.

and the Data attacks. Another one of the most used dataset is KDD-99[31].This data set was used for The Third International Knowledge Discovery and Data Mining Tools Competition, consisting in the feature extracted version of DARPA dataset. The main goal of this competition was to build a system for detecting network intrusions. KDD-99 consists in 4,900,000 single connection vectors each of which contains 41 features and is labeled as either normal or an attack, with exactly one specific attack type between DOS, U2R, R2L, Probing Attack. An evolution of this dataset is the NSL-KDD [31]. It solves some of the problems of KDD-99, like the record redundancy in the training ad in the test set. In fig.2.1 you can see the relation between DARPA, KDD-99 and NSL-KDD. Moreover it is also possible to find other public datasets such as SecRepo [5], a repository in which you can find heterogeneous data sources, like network logs, snort logs and pcap files. Finally the Common Vulnerabilities and Exposure[6] (CVE) is a dictionary of known vulnerabilities, like the recently discovered heartbleed[7]. This dataset is maintained by the MITRE corporation and used in numerous cybersecurity products and services from around the world, including the U.S. National Vulnerability Database.

[5]http://www.secrepo.com *accessed May 2018*
[6] https://cve.mitre.org/ *accessed May 2018*
[7]http://heartbleed.com/ *accessed May 2018*

2.4 Traffic Generators

Traffic generators like SATAN[8] (Security Administrator Tool for Analyzing Networks) and Flame[9] tool can be used to simulate attacks for testing purpose or even to generate web traffic for the learning phase. For example they can be used to obtain pcap files. SATAN is a tool developed to test and find security problems in software and networks. Similarly the Flame tool is a handy tool for evaluating anomaly detection systems operating at the flow level.

[8] http://www.porcupine.or/ *accessed May 2018*
[9]http://www.flame.ee.ethz.ch/ *accessed May 2018*

Chapter 3

Survey on ML approaches for Intrusion Detection

This chapter discusses the state of the art techniques for Intrusion Detection mainly focusing on anomaly based approaches and hybrid solutions due to their capability to recognize unknown attacks. The division of the IDS is done based on the Data Mining and Machine Learning methodologies adopted which are firstly briefly described. For a complete discussion of IDS, including also misuse-based techniques you can look at [6].

3.1 Artificial Neural Networks

Artificial Neural Networks (ANN) make use of neurons for computation. These neurons are structured in several layers, the first layer is the input layer and his output becomes the input to the next one. Finally, the output is fed to the output layer which is the final layer and generates the result of the algorithm. The layers between the input and the output ones are called hidden layers.
As the number of hidden layers increases the learning time of ANN also increases however with the advent of other techniques such as Recurrent Neural Networks (RNN) as well as Convolutional Neural Networks (CNN) this lim-

itation has been targeted and ANN are gaining more popularity.

Lippman et al. [39] process network sniffed data from DARPA dataset, containing the bytes transferred to and from the victim through telnet sessions to count the number of occurrences of a keyword eg., “password”, “permission denied” etc. Then these statistics are given as an input to two ANNs. The first ANN computes the probability of an attack while the second ANN, a Multi-Layer Perceptron (MLP), classifies the network activity into already known attacks thus providing the name of an attack as an output. The software used for this purpose is LNKnet [44], a pattern classification software.The authors obtained results with 80% detection rate and low false positive rate (1/day) while passing from simple keyword selection to keyword selection with dynamically added keywords and discriminative training.

Y. Mirsky et al. [1] proposed Kitsune, an unsupervised IDS which uses autoencoders to differentiate between normal and abnormal traffic patterns. If the traffic pattern is an attack then it provides the class of the attack (if known). They tested their proposal against Gaussian Mixture Model [14] and pcStrem2 [3] showing that Kitsune performances depend on the number of inputs per autoencoder. In any case it out performs both algorithms in terms of Area Under the Curve (AUC) and Equal Error Rate (EER).

In [8] Kim et al. apply Long Short Term Memory (LSTM) to Recurrent Neural Networks (RNN). They train the model on KDD-99 dataset, developing two experiments, in the first phase they try to find the optimal hyper-parameter while in the second they evaluate the model with the previously obtained hyper-parameter. This method result in an average detection percentage of 98.8%, while the false positive rate in around 10%.

3.2 Association Rule Mining

Association rules were firstly proposed by Agrawal et al. in [43] as a method to find co-occurrences in supermarket shopping, also called as market basket analysis. The basic idea of this approach is to find previously unknown rules from data, where a rule is in the following form:

IF *condition(s)* THEN *result*

Indicating that if some element is present (specified in the condition) also the resulting one will be present. The limitation of this approach is that it only works with binary data, while in the reality many problems need more complex data representations. For this reason Fuzzy Association Rule Mining has been proposed by Kuok et al. [42]. This new rules are defined in the form of:

IF *X is A* THEN *Y is B*

Where, roughly, A and B are indicating some possible value that X and Y can assume.
An example in the context of cyber security could be:

IF *NumConntections is* $>$ *100000* THEN *ConnectionType is DOS*

Tajbakhsh et al. [30] applied fuzzy association rules on KDD-99 dataset. To build a classifier fuzzy association rule-sets are exploited as descriptive models of different classes. This method resulted in a nice detection ratio in comparison to similar methods, but his capability to detect new kind of attack is not so effective, anyway it shows 80% detection rate and a false positive rate of 2.95%.

In [27] Ding et al. proposed an extension of Snort[1], an open source misuse IDS which analyzes the network traffic applying rules to identify malign behaviors. The proposed solution is an hybrid approach which uses Snort for

[1]https://www.snort.org/ *accessed May 2018*

the misuse part and association rules to develop anomaly detection. The normal behavior is constructed using Frequent Episode Rules, assuming that the frequent events come mostly from normal interaction with a system. From the anomaly detection part they obtain new attack patterns which are than converted into snort rules, for a more efficient and real-time detection. The resulting detection rate was between 92.2% and 95.7%

3.3 Clustering

As we had seen in 1.4.1 Clustering is an unsupervised method for finding patterns in high-dimensional unlabeled data. The basic idea is to group together data based on some similarity measure.
The idea in the context of intrusion detection systems is that ideally a clustering method should divide the data in two clusters, one with all the normal connections, and the other one with all the attacks.

Almalawy et al. [4] proposed and IDS for SCADA systems. Their approach is based on the *outlier* concept. They cluster the SCADA system in dense clusters using DBSCAN and the resulting n-dimensional space presents some noise data (the outliers) which represent the critical states. The "outlierness" of a state is evaluated through a cut-off function. Moreover they adopt a technique of automatic proximity-based detection rules extraction which enables the monitoring of the criticality degree of the system. Finally Almalawy et al. proposed a method to reduce the high false positive rate, consisting in the re-labeling of the identified critical states by computing the Euclidean distance from each critical state to each normal micro-cluster centroid. If the critical state is located within any normal micro-cluster, it is re-labeled as normal and assigned to that normal micro-cluster. This complex method result in an average accuracy of 98%, but the false positive rate remains quite high, with an average of 16%.

In [12] Lin et al. present a novel feature representation approach: Cluster center and Nearest Neighbor (CANN). In CANN approach two distances are measured and summed, the first one based on the distance between each data sample and its cluster center, computed using k-mean, and the second distance is between the data and its nearest neighbor in the same cluster. Then, this new and one-dimensional distance based feature is used to represent each data sample for intrusion detection by a k-Nearest Neighbor (k-NN) classifier. The evaluation of this method has been made on KDD-99 datasets selecting 6 and 19 features from the original dataset. With the 6-dimensional dataset CANN provides an accuracy of 99.46%, detection rate of 99.28%, and false positive rate of 2.9%.

3.4 Decision Trees

Decision trees are often used in classification problems. They are characterized by a tree structure where leaves represent class, internal nodes represent some test, the related outcomes are represented with the outgoing branches, and paths from the root to one leaf represent a classification rule. An exemplar is classified by testing its features values against the nodes of the decision tree. A simple example of decision tree is represented in figure 3.1.

Bilge et al. [23],[17] introduced EXPOSURE, a system that employs large-scale, passive DNS analysis techniques to detect domains that are involved in malicious activity. They adopted the Weka J48 decision tree program as classifier using 15 features extracted from the DNS traffic. The results vary considerably depending on the data set but overall, using tenfold cross-validation, the detection accuracy results in 98.5% and the false positive rate in 0.9%.

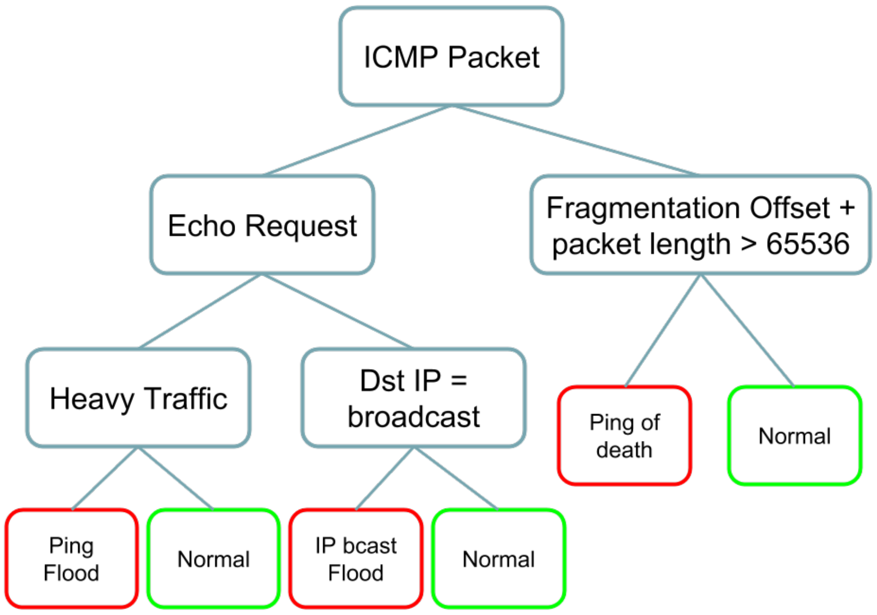


Figure 3.1: A simple example of decision tree for IDS

3.5 Random Forest

Random forest is a classification method which combines decision trees and ensemble learning, a strategy based on the research of better hypothesis, obtained combining simple ones. The forest is composed by many trees that use as input randomly taken data features. Given a forest the resulting prediction can be decided in two different ways: by majority voting or by weighted average.

An hybrid approach in which an anomaly (outlier) detector was employed to feed a second threat classifier (the misuse one) is proposed by Zhang et al. [32].This second one is implemented using proximity between instances. This method has been evaluated on KDD-99 resulting in a 94.7% detection rate and 2% false positive rate for the outlier.

3.6 Evolutionary Computation

Evolutionary computation indicates an ensemble of methods that try to reproduce natural behaviors to solve problems. This methods are Genetic Algorithms (GA), Genetic Programming (GP), Evolution Strategies , Particle Swarm Optimization, Ant Colony Optimization, and Artificial Immune Systems. Both GA and GP are based on the idea of evolution. They start from a random population, than a fitness function is defined to evaluate which elements of the population are the best (for solving the particular problem in exam). The probability to reproduce for an element is proportional to his fitness score, in this way the best elements will reproduce more likely, in a positive trend. The individuals chosen for the reproduction can mix their characterization (crossover) or also go under mutation, which randomly change their characterization. After this phase the individuals with the higher fitness score are used as a new generation and the procedure is repeated. The main difference between GA and GP is the individual description. GA uses bit strings while GP uses entire programs, this involves that the operations of crossover and mutation and significantly easier in the first approach.

In [36] Lu et Traore proposed a rule evolution approach based on GP for detecting known and novel attacks on the network. The initial rules were selected based on background knowledge from known attacks. Each rule can be represented as a parse tree. GP evolves these initial rules to generate new rules using four genetic operators: reproduction, crossover, mutation, and dropping condition operator. This method results in a detection rate near to 100% when the false positive rate is between 1.4% and 1.8%.

Elhag et al. [11] proposed the usage of a Genetic Fuzzy System (GFS), a combination of Fuzzy Association Rules and Evolutionary Algorithms classifiers. This classification scheme is based on a *divide-and-conquer* strategy, in which the original multi-class problem is divided into binary subproblems, which are independently learned by different base classifiers whose outputs are then combined to classify an instance. Moreover they use the OVO binarization that confronts all pairs of classes in order to learn a single model for each couple.The method has been tested on KDD-99, resulting with an accuracy of 99%, an average detection rate of 97.77% and a false positive rate of 0.19 %.

3.7 Support Vector Machine

SVM is a supervised classification method that works on two classes.The method tries to find a separating hyperplane in the feature space such that the distance between the data points of each class and the hyperplane is maximized. Sometimes the sets to discriminate are not linearly separable in the space. For this reason the space should be transformed, making the separation easier, as we can see in figure 3.2 [2] (in our case for example the white dots should represent the normal connections, while the black one the attacks). The transformation is made using particular functions called *kernel functions*.
SVM gives particularly nice results when the number of features is significantly higher than the number of data points.

Wagner et al. [25] proposed to use Support Vector Machine to analyze large volumes of Netflows records. In particular they used a special kernel function, that takes into account both the contextual and the quantitative information of Netflow records. This method results in a nice accuracy of 92% and a false positive rate between 0.004% and 0.033%.

Kim et al. [18] proposed an hybrid IDS using a decision tree for misuse detection and one-class SVM for the anomaly one. The misuse model is used

[2] The figure was taken from
`https://en.wikipedia.org/wiki/Support_vector_machine`

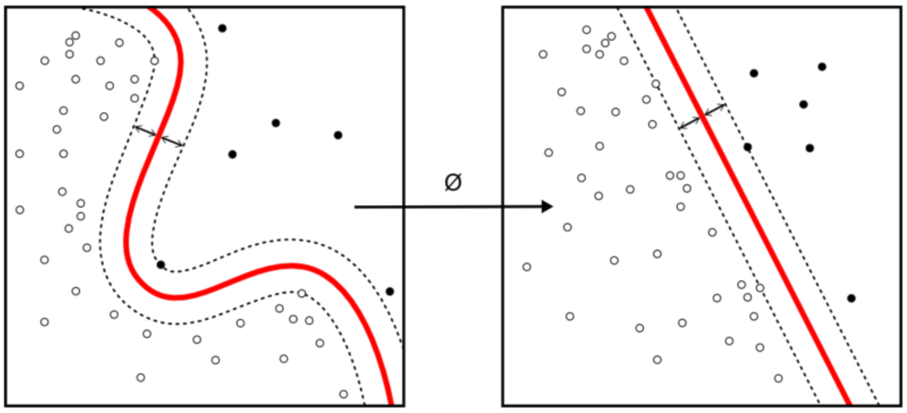


Figure 3.2: A graphic representation of SVM kernel function and separating hyperplane.

to divide the normal data in subsets and each one of them is used to train a one-class SVM classifier for anomaly detection. The idea is that each area for the decomposed normal data set does not have known attacks and includes less variety of connection patterns than the entire normal data set. An anomaly detection model for each normal training data subset can profile more innocent and concentrated data so that this decomposition method can improve the profiling performances of the normal traffic behaviors. The approach has been tested on NSL-KDD distinguishing between known and unknown attacks. The ROC curves show that in the best case the detection rate reaches 90% for previously unknown attacks when the false positive rate is slightly smaller than 10%. For already known attacks the detection rate can reach 100% when the false positive rate remains below 5%.

3.8 Discussion on State of the Art Algorithms

As the purpose of this first phase of work was to asses the state of the art, we have to consider the Accessibility and the Reusability of the presented works, according to the FAIR principles [9]. Unluckily the code is not open source, and even the used datasets are not available. Moreover in the most of

the cases also the documentation leaks of some information, for this reason in chapter 5 only three of the analyzed methods have been reproduced. The most of the approaches are using KDD-99 dataset, but also if it is accessible, they are using a random subset of it, this means that we cannot reproduce the experiments on exactly the same setting. So the first result of this work is to stress the fact that there is the need ti improve the accessibility of the works in this field.
Moreover, as shown in table 3.1 the most of the methods proposed in the state of the art are supervised. This can be a big problem in real world scenarios, as in the most of the cases people don't have labelled data to train their models. The last fact that emerges from this survey is that KDD became a sort of well known "battle ground" used to compare IDS. This is nice, as it gives a common environment to test and compare the methods. But it is also a problems considering the fact that people use only random subsets of the data, and the fact that it is quite aged.

Paper	Year	Method	Data Source
Lippman et al. [39]	2000	MLP	DARPA-98
Kim et al. [8]	2016	LSTM	KDD-99
Tajbakhsh et al. [30]	2009	Fuzzy association rules	KDD-99
Ding et al. [27]	2009	Simple association rules	KDD-99
Lin et al. [12]	2015	K-NN	KDD-99
Bilge et al. [23][17]	2011 - 2014	Decision trees	DNS request
Zhang et al.[32]	2008	Random forest	KDD-99
Lu et al.[36]	2004	GP	DARPA-99
Elhag et al.[11]	2015	GP + Fuzzy association rules	KDD-99
Wagner et al.[25]	2011	SVM	Netflow records
Kim et al.[18]	2014	Decision tree + SVM	NSL-KDD
Y. Mirsky et al. [1]	2018	Auto-encoders	pcap from ip cameras networks
Almalawy et al. [4]	2016	DBSCAN	SCADA systems raw records

🟨 Supervised
🟧 Unsupervised

Table 3.1: IDS state of the art summary.

Chapter 4

Survey on ML approaches for Vulnerability Detection

4.1 Principal Component Analysis

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to reduce the dimensionality of a dataset consisting in many observations of possibly correlated variables into a dataset of values of linearly uncorrelated variables called **principal components**. The first principal component is evaluated preserving the maximum possible variance in the data. The same idea has to be applied to the next principal components, under the constraint that they have to be orthogonal to the previous ones. The vectors obtained in this way will be uncorrelated orthogonal bases set.

In [16] Yamaguchi et al. propose a method for assisted discovery of vulnerability in source code. The idea is to identify API usage patterns and known vulnerabilities to guide code analysis. To do so the source code is mapped to a vector space in the following 4 steps:

- In the first phase, called **API symbols extraction** the source code is tokenized and parsed into individual functions.

- Next **vector space embedding** takes place. Each function is embedded in a vector space in such a way that each dimension is associated with one API symbol.

- This third step consists in the **identification of API usage patterns**. The PCA technique is applied to infer descriptive directions in the vector space, which correspond to *dominant API usage patterns.*

- Last the **assisted vulnerability discovery** phase consists in the expression of each function as a mixture of dominant API usage patterns. In this way the vectorial location of a known vulnerability can be used to identify functions sharing a similar API usage and so possibly containing vulnerabilities.

4.2 Clustering

In [29] Makanju et al. proposed IPLoM (Iterative Partitioning Log Mining) a log data clustering algorithm. The proposed approach works by iteratively partitioning a set of logs. At each step the partition factor is different, in particular the steps are the following: partition by token count, partition by token position and partition by search of bijection. At each step of the partitioning process the resultant partitions come closer to containing only log messages produced by the same line format. When the partitioning phase ends IPLoM produces a line format description for each cluster. This approach is not directly applied to security, but it's clustering capabilities shown some nice result, with a recall of 81% a precision of 73% and a F-measure of 76%.

A more focused approach is proposed by Yamaguchi et al. [16]. They developed an extension of joern[1] (a graph database ad code property graph) to detect taint style vulnerabilities in C programs. Their approach is based

[1]http://mlsec.org/joern/ *accessed May 2018*

	Decision Trees	Linear Regression	Random Forest	Naïve Bayes
Accuracy	72.85%	71.91%	72.95%	62.40%
Recall	74.22%	59.39%	69.43%	29.18%
Far	28.51%	15.58%	23.53%	4.39%

Table 4.1: Reported mean performances in [24]

on the generation of definition graphs and on the clustering of similar callees and types to guide the construction of search pattern for vulnerabilities. The resulting method can detect unknown vulnerabilities, but it also shows some limit, for example it is not able to detect vulnerabilities on concurrent execution, dynamic calls and shared resources like global variables.

4.3 Decision Trees

Chowdhury and Zulkernine [24] present a framework to automatically predict vulnerabilities based on Complexity, Coupling and Cohesion (CCC) metrics. This metrics are important indicators in the software quality which have also been used to detect software faults in general. The authors conducted a large empirical study on fifty-two releases of Mozilla Firefox developed over a period of four years. While they compare 4 ML approaches for the vulnerability prediction: C4.5 Decision Tree, Random Forests, Logistic Regression, and Naïve-Bayes. For all the algorithms Weka with default parameters has been used. The results in table 4.1 show that Decision Trees and Random Forest performed better than the other algorithms.

4.4 Random Forest

In [7] Grieco et al. compared some ML approach to detect memory corruptions in programs binary code. Following this idea they implemented VDiscover, a tool that uses popular ML techniques to predict vulnerabilities in test cases. Starting from analyzing 1039 test cases taken from the Debian Bug Tracker the dataset has been preprocessed in two different ways, with *word2vec* and with *bag-of-words*, while three ML approaches have been tested: Logistic Regression, Multilayer Perceptron (MLP) and Random Forest. In this experiment the method which performed better in terms of accuracy was random forest, trained using dynamical features.

Also Younis et al. [10] take under exam different machine learning approaches, but their purpose is quite different. They try to identify the attributes of the code containing a vulnerability that makes the code more likely to be exploited. So their purpose is not only to identify vulnerabilities, but to check if they are exploitable. The authors examined 183 vulnerabilities from the National Vulnerability Database for Linux Kernel and Apache Http Server, finding 82 to have an exploit. After that, the authors characterized the vulnerable functions with and without an exploit using the selected eight software metrics: Source Line of Code, Cyclomatic complexity, CountPath, Nesting Degree, Information Flow, Calling functions, Called by functions, and Number of Invocations. Apache HTTP server. Only a combination of this metrics has been used in the experiment, in fact the authors used three different feature selection methods: correlation-based, wrapper and principal component analysis. The classification methods used are Logistic Regression, Naïve Bayes, Random Forest, and Support Vector machine. The experiments results (table 4.2) show that the best approach is Random Forest.

In [19] Scandariato et al. propose an approach based on text mining the source code of software components. Namely, each component is characterized as a series of terms contained in its source code, with the associated frequencies. These features are used to forecast whether each component is likely to contain vulnerabilities. Similarly to what has been done in the previous paper the authors tested different ML approaches: Decision Trees,

	Logistic Regression	Naïve Bayes	Random Forest	SVM
Accuracy	60%	70%	76.6%	66.7%
Recall	60%	70%	76%	67%
Far	50%	15%	32%	67%

Table 4.2: Results reported in [10] for classification without feature extraction. for a more detailed results refer to the paper.

k-Nearest Neighbor, Naïve Bayes, Random Forest and Support Vector Machine (SVM). Between these Random forest and Naïve Bayes resulted as the more effective. Both of them have been tested using Weka, with default setting for the Naïve Bayes, while the number of random trees for Random Forest hat been increased to 100. The experiments shown that between these two approaches Random Forest is the one doing better with 82% recall and 59% precision, against 73% precision and 55% recall for Naïve Bayes.

4.5 Support Vector Machine

Saul et al. [26] exploited on-line sources of vulnerabilities data to train a classifier on known vulnerabilities to than detect new ones. The labeled vulnerabilities based on their exploitation and than they extracted a high-dimensional ($d = 93578$) feature vector of binary and integer valued features for each vulnerability. Finally they applied SVM on this feature vectors.

In [13] Perl et al. apply SVM to find possible dangerous code. The authors conducted a large-scale evaluation of 66 GitHub projects with 170,860 commits, gathering both metadata about the commits as well as mapping CVEs to commits to create a database of *vulnerability-contributing commits*. The resulting approach over-performed FlawFinder [2] resulting in a precision of 56% in the best case.

[2]https://www.dwheeler.com/flawfinder/ *accessed June 2018*

Hovsepyan et al. [20] proposed an approach based on the analysis of source code as text, without using additional informations, like code churn. The experiment has been conducted on Java code, which has been transformed in feature vectors. In detail every word in a document has been treated as a feature for that document, and the number of occurrences of that word as the value for that feature. The problem has been treated as a binary classification problem, where a file can be *vulnerable* or *clean*. Using an SVM classifier they obtained an accuracy, precision and recall of 87%, 85% and 88% respectively.

Chapter 5

Comparative evaluation of intrusion detection systems

Many of the surveyed papers report evaluation results in terms of accuracy, detection rate and FAR. This may lead to a comparison between them, however the original experiments were conducted with different settings. Although most of the proposed methods were evaluated against the KDD-99 dataset, each of them used a different random subset of it. Nevertheless, in table 5.1 we report the performance of the methods as they are claimed by the authors to provide an informal comparison between them.

Y. Mirsky et al. [1] is not appearing in the table because the results are not reported in terms of FAR, accuracy or detection rate. As previously said the best result is in terms of Area Under the Curve (AUC) with one input per auto-encoder. Depending on the analyzed dataset their AUC score goes between 0.79499 and 0.99997. As Mirsky et al. explain AUC is the probability that a classifier will rank a randomly chosen anomalous instance higher (in terms of attack) than a randomly chosen normal instance. In other words, an algorithm with an AUC of 1 is a perfect anomaly detector on the given dataset, whereas an algorithm with an AUC of 0.5 is randomly guessing labels.

Table 5.1 is insufficient to provide a formal assessment of existing meth-

	Accuracy	**Detection Rate**	**FAR**
Lippman et al. [39]	-	80 %	1/day
Kim et al. [8]	96.93%	98.8%	10.04%
Tajbakhsh et al. [30]	-	80.6%	2.95%
Ding et al. [27]	-	92.2% - 95.7%	-
Almalawy et al. [4]	98%	-	16%
Lin et al. [12]	99.76%	99.99%	0.003%
Bilge et al. [23][17]	98.5%	-	0.9%
Zhang et al.[32]	-	94.7%	2%
Lu et al.[36]	-	∼100 %	1.4%-1.8%
Elhag et al.[11]	99%	97.77%	0.19%
Wagner et al.[25]	92%	-	0.004%- 0.033%
Kim et al.[18]	-	90%	<10%

Table 5.1: Results reported in the proposed papers in terms of Accuracy, Detection Rate (recall) and FAR.

ods, therefore the next step is to address this issue by re-implementing existing methods and compare them in the same experimental setting. It has to be noticed that in some cases the implementation of the methods may not be exactly the same as it was in the original proposal, for many reasons, e.g. the lack of implementation details in the paper, lack of access to the original code.

5.1 Experimental setting:

All the experiments have been conducted using scikitLearn, TensorFlow, Keras (with GPU support) and Weka. For reproducibility reasons the characteristics of the environment in which the experiments have been conducted are summarized in the following lines:

- Calculator:
  - OS: Ubuntu 16.04 LTS
  - CPU: Intel Core i7-6500U 2.5GHz x 4
  - RAM: 12 GB
  - ROM: 55 GB
  - GPU: GeForce 920M/PCIe/SSE2
- Software versions:
  - CUDA: 9.0.176
  - CUDNN: 7.1.4
  - Python: 2.7.12
  - Tensorflow: 1.8.0
  - Keras: 2.2.0
  - Scikit Learn: 0.19.1
  - Weka: 3.8.2

5.1.1 Dataset

The choosen dataset is NSL-KDD. As previously explained in 2 it derives from KDD, a very known and used dataset for IDS. The dataset is composed by training and testing dataset, in which each item is composed by 41 features (table 5.2) extracted from DARPA. For more detail about the feature refer at [31]. Of course this dataset is quite dated, but it still can be applied as an effective benchmark dataset to compare different intrusion detection methods.

NSL-KDD solves many of the problem of the KDD-99 dataset:

- It does not include redundant items in the training dataset and neither in the testing one.
- The number of records in the train and test sets are reasonable, which makes it affordable to run the experiments on the complete set, without the need to randomly select a small portion of it.

While the training dataset contains 22 attacks types, divided in 4 attack families, the testing one adds 17 new attacks types. This characterization makes possible to test the IDSs capability to detect new attacks, a very important aspect for an IDS. The attacks types and the related attack category present in the dataset are described in table 5.3.

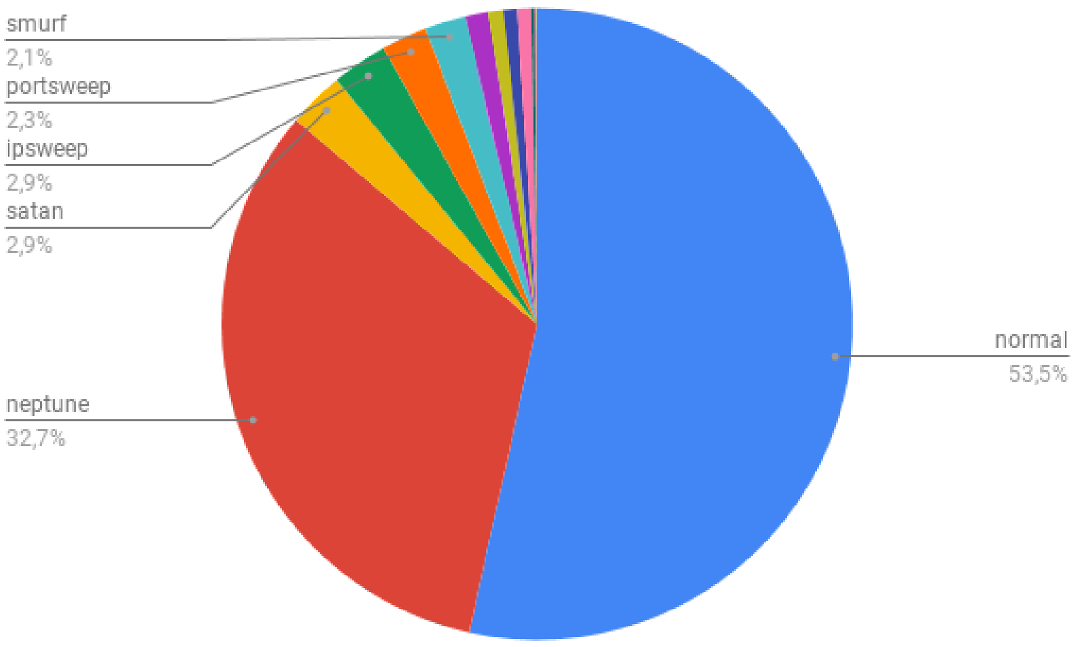
In figure 5.1 you can find the pie charts describing the attacks percentage in the training and in the testing dataset, while in figure 5.2 you will find similar charts about the attacks categories.

Beyond this categorization we can see that the train and the test sets are composed by 53.5% and 43.1% normal items respectively. This is probably the most important categorization we need to care of, because many approaches use anomaly detection, which has to detect new attacks and of course they cannot give a name to something they are seeing for the first time. For this reason in this work the problem is approached as a Binary Classification one, in which the task is to discriminate between *normal* and *malign* connections.

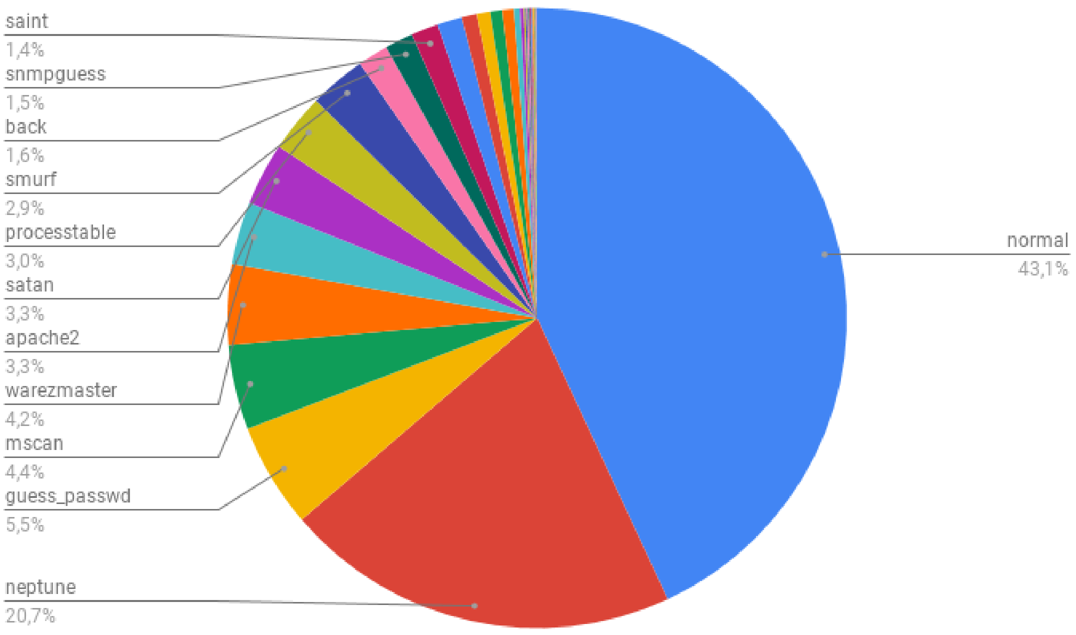
Basic features	
1	duration
2	protocol_type
3	service
4	flag
5	src_bytes
6	dst_bytes
7	land
8	wrong_fragments
9	urgent
Content features	
10	hot
11	num_failed_logins
12	logged_in
13	num_compromised
14	root_shell
15	su_attempted
16	num_root
17	num_file_creations
18	num_shells
19	num_access_files
20	num_outbound_cmds
21	is_host_login
22	is_guest_login

Traffic features	
23	count
24	srv_count
25	serror_rate
26	srv_serror_rate
27	rerror_rate
28	srv_rerror_rate
29	same_srv_rate
30	diff_srv_rate
31	srv_diff_host_rate
Host Traffic Features	
32	dst_host_count
33	dst_host_srv_count
34	dst_host_same_srv_rate
35	dst_host_diff_srv_rate
36	dst_host_same_src_port_rate
37	dst_host_srv_diff_host_rate
38	dst_host_serror_rate
39	dst_host_srv_serror_rate
40	dst_host_rerror_rate
41	dst_host_srv_rerror_rate

Table 5.2: NSL-KDD features grouped by their domain. [34]

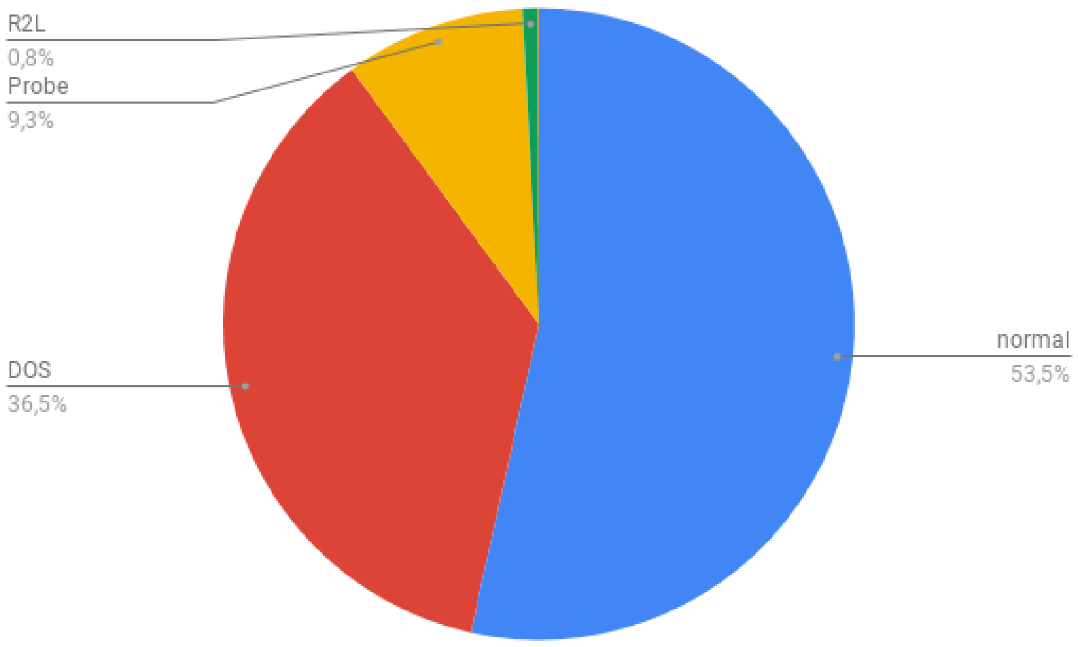


(a) Attack types in the training set.

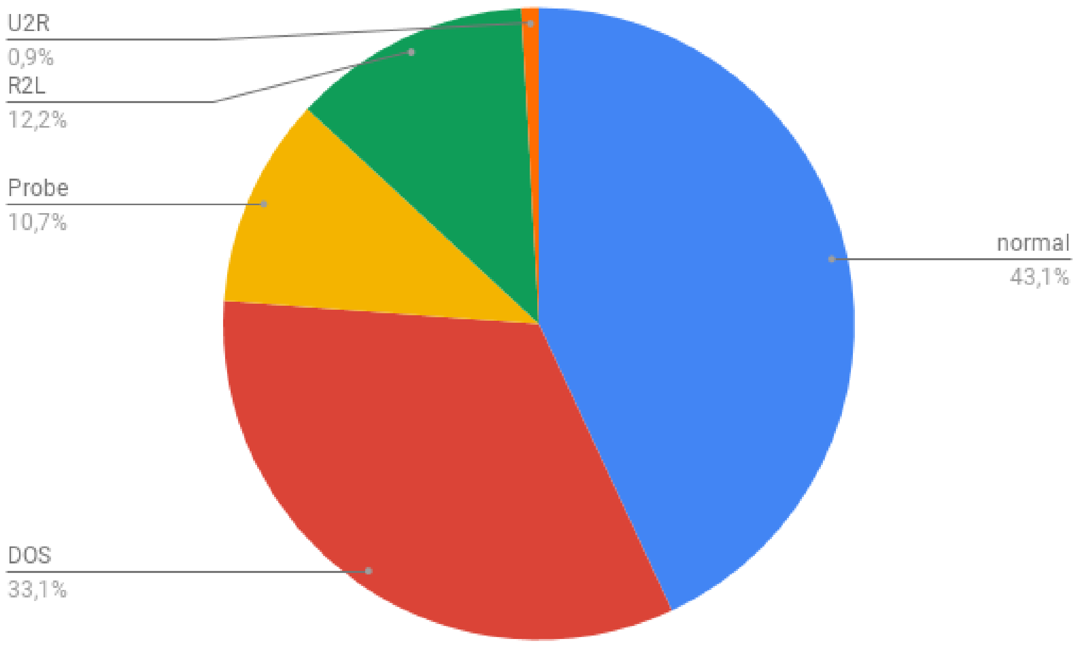


(b) Attack types in the testing set.

Figure 5.1: NSL-KDD attacks-types pie charts.



(a) Attack categories in the training set.



(b) Attack categories in the testing set.

Figure 5.2: NSL-KDD attack categories pie charts.

	Attacks in training set	**Additional attacks in testing set**
DOS	back, neptune, smurf, teardrop, land, pod	apache2, mailbomb, processtable, udpstorm
Probe	satan, portsweep, ipsweep, nmap	mscan, saint
R2L	warezmaster, warezclient, ftp_write, guess_passwd, imap, multihop, phf, spy	sendmail, named, snmpgetattack, snmpguess, xlock, xsnoop, worm
U2R	rootkit, buffer_overflow, loadmodule, perl	httptunnel, ps, sqlattack, xterm

Table 5.3: Attack types divided by category in NSL-KDD training and testing dataset.

5.2 Methods Reproduction

As previously said one of the purposes of this work is the assessment and the comparison of the method presented in the state of the art.
For this reason this section is going to illustrate in detail some of them, explaining how they have been reproduced and which are they results. The reproduced methods are three, those best documented, and therefore reproducible:

- Hybrid random forest [32]
- Cluster center + K-nearest neighbor [12]
- Decision tree + One-Class SVM [18]

The reproductions are not always strictly faithful, because the code of the described methods is not available, and as it often happens, their description is not enough detailed to understand all their aspects. Moreover in some case the authors performs a different kind of classification, for example multi-class instead of binary and so on. As explained before the reproduction has to be done on a common environment to give sense to a comparison, this includes

also the task, which in this work is binary classification between normal and malign connections.
For this reasons the results shown in the following paragraphs will be different from the one reported in the original papers. Moreover all the experiments made for this work have been done using the NSL-KDD train dataset for training and the test one fore testing. This can seem obvious, but in some case the authors used a partition of the training dataset for the evaluation of their methods. This lead to higher performances because the testing data set contains zero days attacks, which are harder to be recognized.

5.2.1 Hybrid Random Forest

Zhang et al. [32] proposed an Hybrid IDS based on random forest. Their method consists in the combination of a misuse based classifier, followed by an outlier one. For all the three methods, misuse based, anomaly based (outlier based) and the hybrid one the authors made some experiment using KDD-99 searching the parameters which were optimizing their classifiers performances. In the following reproduction these same parameters have been used.

Misuse detection: This step is the simplest one and consists in the application of a random forest classifier. The classifier has been configured to use *mtry* 15 and 100 trees. Moreover the features 6, 20 and 21 (tab 5.2) have been excluded. The misuse classifier shown the following performances:

- Accuracy: 77.64%
- Precision: 94.51%
- Detection rate (recall): 64.47%
- FAR: 4.94%

As we can expect from a misuse method tested with zero-days attacks the detection rate is not very high, while the false alarm rate is quite low.

Anomaly detection: For the anomaly detection the authors proposed to construct the normal patterns categorizing the network traffic by service. For this reason they use a random forest classifier using the service as target class and excluding all the *malign* connections. In this way the normal patterns are identified for each service and the trained model can be used for outlier detection. In this first step the random forest classifiers uses *mtry* 15 and 10 trees.

After the normal service patters are identified the authors proposed two types of outliers for the outlier detection. First all the connection which are classified wrong in term of service are marked as outlier (so as attack). Then for the other an outlier-ness score is evaluated, considering as outlier all the connections that falls over a predefined threshold. As described in the paper the outlier-ness can be calculated over proximities.

class(k) $= j$ denotes that the item k belongs to class j.

prox($n,\ k$) denotes the proximity between items n and k. The proximity between two items is computed as the number of trees in which the items are classified in the same leaf. The average proximity from item n, belonging to class j, to case k (the rest of the items belonging to the same class) is computed as

$$P(n) = \sum_{class(k)=j} prox^2(n,k)$$

Denoting with N the number of cases in the dataset, the raw outlier-ness of item n has been defined as

$$\frac{N}{P(n)}$$

For each class, the median and the absolute deviation of all raw outlier-ness are calculated. The median is subtracted from each raw outlier-ness. The result of the subtraction is divided by the absolute deviation to get the final outlier-ness.

The only difference from the method described in the paper is that, instead of constructing the normal patterns for the network service, here they have been done fore the protocol types. The reason is very simple, and it is that in this scenario the results were better using the protocol types. Moreover, as the threshold over which an outlier-ness score was considered to be determinant for an anomaly, the experiment have been repeated with many different thresholds. The best one are as follows:

- Accuracy: 84.08%
- Precision: 81.86%
- Detection rate (recall): 92.56%
- FAR: 27.11%

As expected the detection rate increased from the misuse detection, but the false alarm rate is worst.

Hybrid detection: This last classifier combines the two previous ones. It applies as first the misuse classifier, and than the anomaly one on all the items that are classified as normal by the misuse detection. For this step the authors set the misuse random forest to use 15 trees, *mtry 34* and to ignore the features 2,7,8,9,15,20 and 21. While for the anomaly detection the number of trees is 35 and the value of *mtry* is 34. The authors proposed to use as threshold 1%, this means that only the 1% of the items with the higher outlier-ness will be marked as anomaly. In this experimental evaluation the used threshold has been reduced up to 0.12%, which gave the best results in terms of FAR. Moreover it has been tested both with the original misuse detection parameters and with the newly proposed. The best results have been obtained with the original settings for the misuse detector. Increasing the threshold some better result in terms of detection rate can be obtained, even if the FAR increase.
In table 5.4 you can see how increasing the threshold increases the detection rate (recall) but it increases also the FAR, this happens because increasing the threshold means considering as attacks also connections which are more similar to the normal ones. In this case probably the best result is the one

Threshold	Accuracy	Precision	Recall	FAR
0.12%	77.64%	93.14%	65.52%	6.33%
20%	84.62%	90.02%	82.08%	12.03%
50%	83.37%	80.97%	99.42%	30.87%

Table 5.4: Performances of the Hybrid Random Forest IDS when the threshold changes.

with threshold set to 20% which has a nice detection rate and the false alarm rate remains not too high.

5.2.2 Cluster Center and Nearest Neighbor

Cluster Center and Nearest Neighbor (CANN) is a method proposed by Lin et al. [12]. It is the only one that tries to manage the data set, trying to find a more representative feature to be used for the classification.
As first K-mean is applied to find clusters and cluster centers, setting the number of cluster to be found to the number of classes present in the classification problem. After this, for each item in the dataset two distances are computed:

- the distance to each one of the cluster centers;
- the distance to the nearest neighbor of the item in the same cluster it belongs to.

These two distances are then summed and used as new unique feature for a k-NN classifier. The reason to use this new one-dimensional dataset is very easy to understand, working on a one-dimensional dataset instead on one with 41 dimensions is less expensive.
Considering the item D_i in figure 5.3 his new feature is computed as

$$Dis(D_i) = \sum_{j=1}^{5} Dis(D_i, C_j) + Dis(D_i, N_1)$$

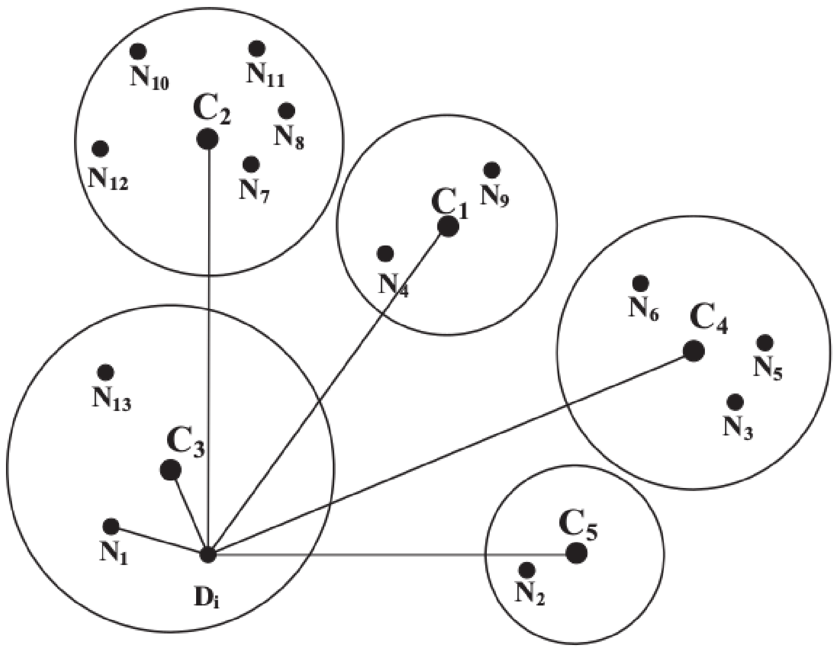


Figure 5.3: CANN: new feature computation.
source: [12]

where C_j is the cluster center of the *j-th* cluster and N_k is the *k-th* nearest neighbor of the item D_i. As we are considering only the nearest neighbor k is 1, but it should be increased using a sum also in the second part of the equation as follows:

$$Dis(D_i) = \sum_{j=1}^{5} Dis(D_i, C_j) + \sum_{k=1}^{K} Dis(D_i, N_k)$$

In the paper Lin et al. use 5 cluster, as they are classifying over the attack families, while in this work the task is to discriminate between normal and malign connections, so the number of cluster is only two. Moreover they authors used KDD as dataset, considering only a subset of his features, in particular the use two different datasets, one composed by 6 features and one composed by 19 features. Referring to table 5.2 the used features are:

- 6-dimensional dataset: 7, 9, 11, 18, 20 and 21;
- 19-dimensional dataset: 2, 4, 6, 12, 23, 25, 26, 27, 28, 29, 31, 32, 33, 34, 35, 36, 37, 38 and 39.

Testing this approach the best results have been obtained using the 19-dimensional dataset and k=1 and 2 for the k-NN classifier over the new one-dimensional data while working with 10-fold cross validation and on the test set respectively. As we can expect using 10-fold cross validation on the training data the results are strongly better than testing it on the test data, because this is not an anomaly detection method, so it is not able to find zero-days attacks. This is proved by the results reported in table 5.5. This results may look very bad, but as already said we should consider that the new feature has not been used for anomaly detection, so they are normal. The new one-dimensional dataset should be used to train an anomaly-detection model, instead of a simple k-NN, to understand if it could improve his performance.

	Accuracy	**Precision**	**Recall**	**FAR**
10-fold	88.05%	93.48%	79.9%	4.86%
test-set	47.60%	52.92%	73.54%	86.47%

Table 5.5: Results of CANN using 10-fold cross validation and the NSL-KDD test set.

5.2.3 Hybrid Detection combining Decision Trees and one-class SVM

Kim et al. [18] proposed an hybrid IDS in which a C4.5 decision tree is used for misuse detection. The tree is also used to decompose the normal training data into small subsets which are then used to train multiple one-class SVM models (with Gaussian kernel) for the anomaly detection. As described in the paper the training phase is composed by the following steps:

1. Prepare a training data set consisting of normal data and known attack data.
2. Build a misuse detection model using a decision tree algorithm based on the training data set.
3. Decompose the normal training data into subsets according to the decision tree structure, data in the same leaf belong to the same subset.
4. For each normal leaf of the decision tree, build an anomaly detection model using the 1-class SVM algorithm based on a normal data subset for the leaf.

This means that if the decision tree marks an item as an attack, it is actually considered to be an attack, while if it is marked as normal it will also be subject to anomaly detection (as we can see in figure 5.4). The one-class SVM are used as anomaly detector as they model a class pattern and then they can be used to test the belonging of new items to this class (in our case normal connections).

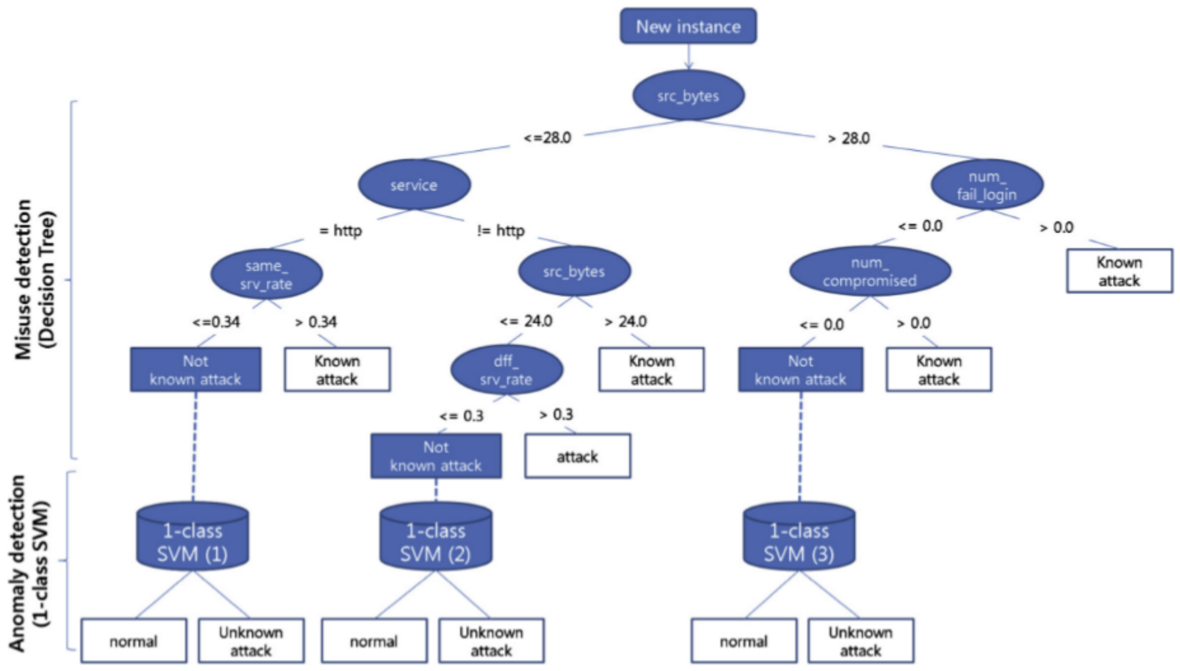


Figure 5.4: Diagram of the hybrid IDS based on decision trees and one-class SVM.
source: [18]

While reproducing this method a different decision tree algorithm has been used, as *sklearn* does not implement the c4.5 one. In particular a classification and regression tree (CART) has been adopted. For the one-class SVM two parameters are important, the parameter γ (gamma) which is characteristic of the kernel function and in some sense models the capability of the model to detect new attacks, and the parameter ν (nu) which controls the fraction of training instances that are allowed to be rejected (and so not considered as belonging to the modeled class).

Reproducing the method three different values of γ have been tested: 0.01, 0.1 and 1. While for ν 0.01 and 0.5 have been chosen.
Regarding the decision three the minimum number of instances per leaf has been set to 0.1% of the dataset size.
The results obtained applying only misuse detection (the simple decision tree) are the following:

- Accuracy 82.12%
- Precision 86.53%
- Recall 81.24%
- FAR 16.71%

This results are not bad, but the false alarm rate is quite high for a misuse detector. this means that a possible way to improve the method is trying to understand how to decrease this high FAR.
Moreover we can expect a FAR grater to this one for the hybrid detection, because it usually suffers for high false positives and in this method the items marked as attacks by the decision tree are not even evaluated by the anomaly detection, so for sure the FAR won't decrease. In table 5.6 the results obtained by the hybrid detection while the parameters γ and ν change are reported. It is possible to observe that when ν is 0.5 the detection rate (recall) increase, but this is too much expensive in terms of FAR.

ν	γ	Accuracy	Precision	Recall	FAR
0.01	0.01	85.62%	87.07%	87.77%	17.22%
	0.1	85.75%	87.02%	88.1%	17.36%
	1	86.84%	86.82%	90.64%	18.18%
0.5	0.01	74.31%	69.8%	96.72%	55.31%
	0.1	74.2%	69.66%	96.85%	55.74%
	1	75.53%	70.4%	98.38%	54.66%

Table 5.6: Results of the Hybrid IDS based on decision tree and one-class SVM while γ and ν vary.

Setting ν to 0.01 the detection rate increase from the misuse model, while the FAR increases only by a small percentage.

5.3 Comparison Summary

From the preceding experiments we can observe that hybrid approaches can achieve nice results. In detail, compared with the same settings, using One-Class SVM on subsets of the original data, identified by a decision tree classifier outperforms the other methods, as shown in table 5.7. In the table F1 measure has been added, lo let the reader compare the results easily. Probably the reason behind this results is the choice of using a decision three to split the data, before applying outlier detection. In fact it looks quite intuitive that if we want to model a description of a class, the more the used items are similar to each other, the better the resulting description will be.

Method	**Accyracy (%)**	**Precision (%)**	**Recall (%)**	**FAR (%)**	**F1 Score (%)**
Hybrid Random Forest	84.62	90.02	82.08	12.03	85
Cluster Centers + Nearest Neighbor	47.60	52.92	73.54	86.47	61
Decision Trees + One-Class SVM	86.84	86.82	90.64	18.18	89

Table 5.7: State of the art IDSs experimental comparison summary.

Chapter 6

A real world Scenario and a novel unsupervised approach to intrusion detection

Until this point only *positive* scenarios have been treated. In real world the most of the time data is raw, and not labelled. In this work we will take as Real World Scenario: the HAPS (Holistic Attack Prevention System) project. The aim of this project is to take many different logs (web server, application and so on) and apply Machine Learning techniques to detect cyber security threaths. Working on raw log files is a different and harder work that starting from a dataset designed to be used for Machine Learning as the KDD one. In fact these logs are completely unlabelled and they need to be preprocessed in order to apply ML methods. Moreover for different Log types a different preprocessing needs to be applied, and the same stands for the machine learning technique. For this reason this works focuses only on web server logs.

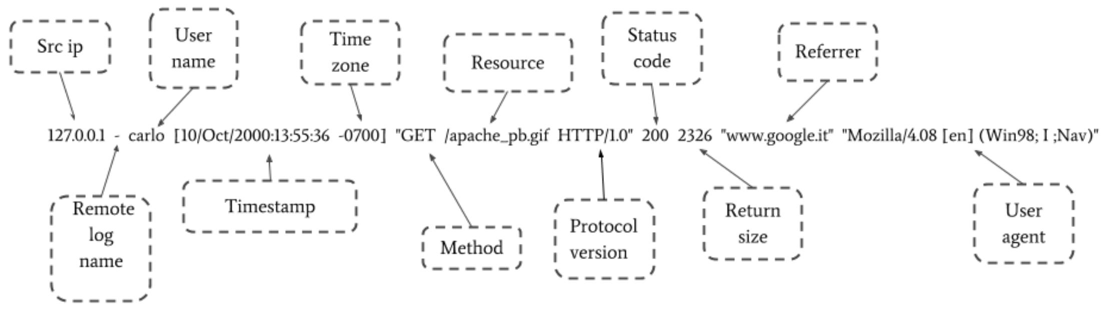

Src ip
User name
Time zone
Resource
Status code
Referrer
127.0.0.1 - carlo [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326 "www.google.it" "Mozilla/4.08 [en] (Win98; I ;Nav)"
Remote log name
Timestamp
Method
Protocol version
Return size
User agent

Figure 6.1: Combined Log Format example.

6.1 Web Server Logs Analysis

6.1.1 Structure

The web logs analyzed in this work are produced by an Apache Web Server and they are expressed in Combined Log Format (fig. 6.1). In the following lines the different fields of this format are explained.

- **Src ip:** It indicates the IP address of the client which generated the request.
- **Remote log name:** It represents the client identity, determined by *identd* on the client machine. If '-' is supplied instead of a valid string it idicates that this information is not available. Moreover the Apache documentation suggest not to use this field because it is not reliable.
- **User Name:** The user id of the person which requested the web resource, as established by the HTTP authentication.
- **Time stamp:** Date and hour of the moment in which the request has been received by the server.
  It is expressed as [*day/Month/year:hour:minutes:seconds*] where:
  - *day* is expressed by 2 digits;
  - *Month* is expressed by 3 characters with the first capitalized;

  - *year* is expressed by 4 digits;
  - *hour* is expressed by 2 digits;
  - *minutes* is expressed by 2 digits;
  - *seconds* is expressed by two digits.

- **Time zone:** It represents the time zone related to the previous filed. It can begin with + or -, depending on the zone.
- **Method:** The request method.
- **Resource:** The requested resource in form of server path. It can be followed by the request parameters.
- **Protocol version:** It indicates the HTTP protocol version.
- **Status code:** The status code that the server returns to the client.
- **Return size:** The size (in bytes) of the object returned by the server, excluding the headers.
- **Referrer:** The web page in which the client was when it made the request. For example, following a link from the web site of your company, the resulting request will have the url of your company web site as referrer field. This filed can be empty (-) if the request is generated by directly writing an url in the url bar, or when the field is intentionally leaved empty, as it often happens when the request is generated by a bot.
- **User Agent:** It contains information relative to the client browser and operating system.

6.1.2 Collected Data

The log files analyzed in this work cover a period of 30 days, for a total of 627.180 requests. A detailed analysis has been conducted using GoAccess[1],

[1]https://goaccess.io/

a terminal based log-analyzer. This analysis shows that the most of the connections come from bots, as it can be derived from many different aspects. As first the 8.83% of the static request are directed to *robot.txt*, a particular file used by bots to discover which part of the web site are forbidden to them. Second, almost the 83% of the request come from North America, which is a quite strange behavior considering that the website in object is Italian. Moreover more than the 83% of the request comes from declared web crawler and another 8% from unknown browsers (which are generally related to bots). Finally the traffic distribution along the day is almost constant while it became more varied excluding the web crawlers (as shown in figure 6.2).

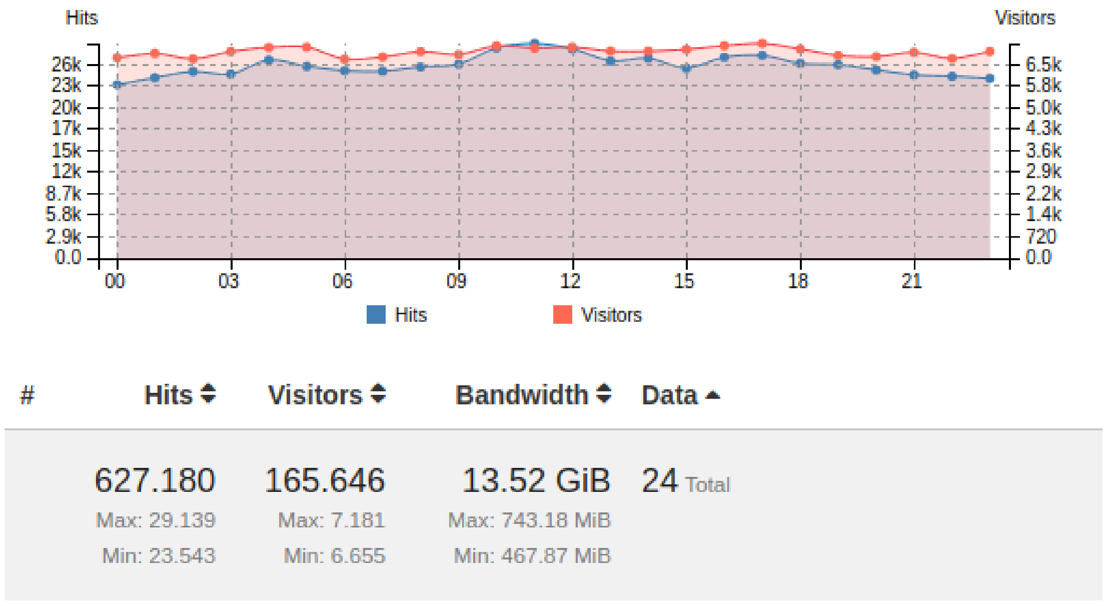
It has also to be considered the fact that GoAccess cannot detect all the bots, because in many case they try to hide their identity, this means that the percentages shown before can even be worst.
Another important aspect of the analyzed logs is that the remote log name and the user name fields have been obfuscated by the web site owner for privacy reasons.

6.2 Data Preprocessing and Feature Extraction

This phase of the work is one of the most important, because the quality of the final ML model is strongly dependent to the quality of the data representation.
As previously said the data under exam are completely unlabelled. For this reason three possible high level approach can be applied:

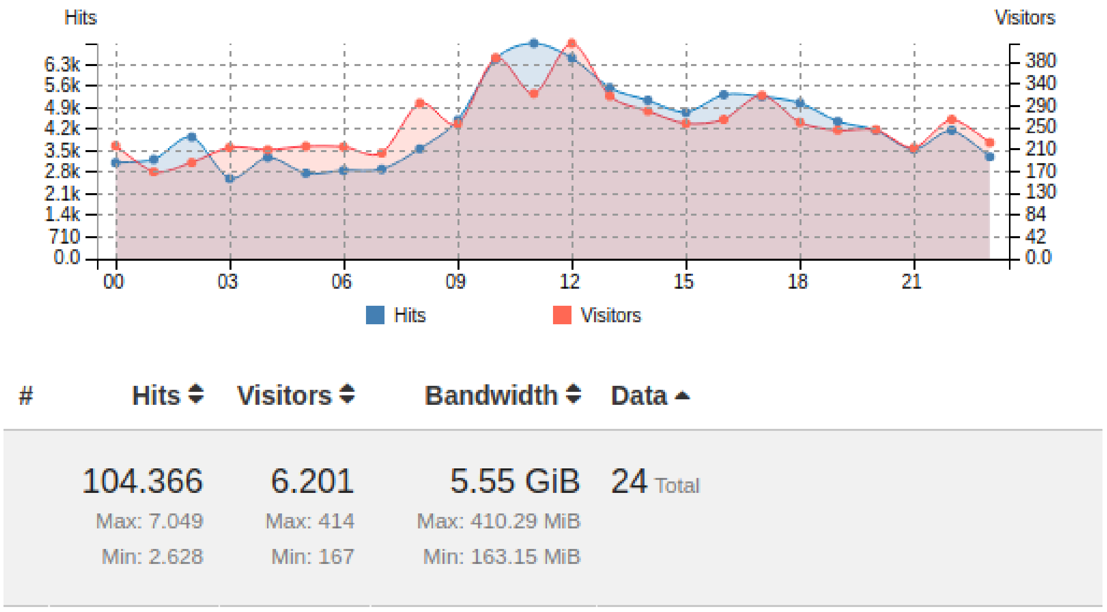
- Use some completely unsupervised techniques (clustering)

- Apply outlier/anomaly detection starting from the assumption that the most of the training data are normal.



#	Hits ⬍	Visitors ⬍	Bandwidth ⬍	Data ▲
	627.180 Max: 29.139 Min: 23.543	165.646 Max: 7.181 Min: 6.655	13.52 GiB Max: 743.18 MiB Min: 467.87 MiB	24 Total

(a) With web crawlers.



#	Hits ⬍	Visitors ⬍	Bandwidth ⬍	Data ▲
	104.366 Max: 7.049 Min: 2.628	6.201 Max: 414 Min: 167	5.55 GiB Max: 410.29 MiB Min: 163.15 MiB	24 Total

(b) Without web crawlers.

Figure 6.2: Traffic distribution with and without web crawlers. The data field indicates the number of hour considered.

- Apply rules to the data to find labels and then use some classification technique.

The first approach should be a bit much undetermined, because we don't have enough information to discriminate between normal and attack logs simply clustering them together. The third one instead consists essentially in misuse detection, which, as you already know at this point, is not able to detect new attacks. Moreover starting from a set of rules to apply labels to the data the resulting classifier will simply be a model representing these same rules, nullifying the sense of making the model itself. As emerged in chapter 3 anomaly detection can be a very effective technique to detect known and new attacks. Moreover, if the outlier technique is used there is no need to start from labelled data, but the starting assumption can be relaxed to the need of training the normal models from a datasource consisting of mostly normal items (as shown in sec. 5.2.3). For these reasons this work focuses on the outlier technique, which also emerges in many works related to the web log analysis as [38], [21], [35].
The reason why this choice is treated at this point is that the data preprocessing step can depend on the model that is going to be realized. For example we said that we need to assume that the most of the data is made by normal items, but we also said that our data is completely unlabelled.
In some case it can look quite realistic to assume that the most of the traffic of a web server is normal, but, to improve the truthfulness of this assumption the first step of the data preprocessing phase has been to define a set of heuristics to filter some known attack fingerprint. These heuristics are directly applied to the raw logs. In detail they look for fingerprints in the following fields:

- Resource, e.g. request containing one ore more ../ are often used by attacker to access file that are outside of the website itself.

- Referrer and user agent, e.g. SSI tags <!– –> are often inserted by an attacker which tries to execute some code.

- Status code, e.g. even if it is not a rule a *5xx* status code can be the consequence of an attack to the web service.

As an attacker can exploit a very large number of different techniques also the number of fingerprints to search for is very huge, for this reason they are not listed here, but you can refer to [45], [46]. As explained in these works not all the fingerprints described are enough to consider the connection dangerous, but in some case they are related to other ones, so also this aspect is considered when the data is pre-filtered. Obviously this set of heuristics does not cover all the possible attacks, but it is only a simple way to pre-filter the data. Moreover these heuristics can be updated in any time to obtain a more effective filter.

Beside this first filtering phase the data still needs to be preprocessed in such a way that it can be managed by a ML algorithm.
Analyzing the possible attacks it has appeared that probably a single model should not be enough to catch all the possible attacks types. For example there are attacks with are more related to the single connection structure, like requests to *root.exe*; while other are related to a set of connection, usually sessions, like DOS attacks.
For this reason there should be the need to evaluate more models, and so more datasets, in such a way to analyze both the single connections and the sessions.

6.2.1 Single Connections

The first approach to represent the single connections is to simply split them in their fields, which will be used as features in the dataset.
This approach can be useful for some aspect, but is probably too simplistic, there is the need to understand which aspects can be more relevant to individuate an attack.
A significant part of a connection is the requested resource and the parameters associated to it. For this reason the parameters themselves can be used

IP		Att1	Att2	Att3	N. Att.
192.168.1.4	...	"Carlo"		5	2
192.168.1.2					0

Table 6.1: Single connections dataset example.

as feature in the dataset, and the value associated to the feature will be the parameter value.
Since not all the requests use all the parameters, the not used ones will be left empty in the dataset. Another useful information can be the number of parameters passed to the request.
Let's imagine to have to represent a log file containing only two requests. Both of them pointing to the same resource */r1.html*. This resource is related to three different parameters: *att1, att2* and *att3*.

```
192.168.1.4 - - [10/Oct/2000:13:55:36 -0700] "GET /r1.html?att1="Carlo"&att3=5 HTTP/1.0" 200
2326 "www.google.it" "Mozilla/4.08 [en] (Win98; I ;Nav)"

192.168.1.2 - - [10/Oct/2000:13:55:36 -0700] "GET /r1.html HTTP/1.0" 200 2326 "www.google.it"
"Mozilla/4.08 [en] (Win98; I ;Nav)"
```

The resulting dataset should be as shown in table 6.1

However this kind of representation will result in a high increase in the number of features, because for each single connection in the logs we will have a feature for each possible parameter of each different resource ever requested in the whole log.
Moreover, with the perspective of applying outlier detection, the most important thing is to have the capability to define clearly the normal models. But the normal usage pattern will be different for each resource. So to solve both these problems the whole dataset can be splitted by resource. In this

way each dataset will have less features, representing only the parameters associated to the resource considered (reducing also the computation time). And it will be possible to create a different model for each resource, having like this a more ”fitting” model to the actual usage pattern of that resource.

6.2.2 Sessions

To reason at this level, the first thing that has to be done is to reconstruct the sessions. Even in this case this purpose is followed using an heuristic. Similarly to what has been done in [37], [15] the chosen heuristic consists in grouping connection inside a session if hey have common IP and User Agent. Moreover all the connection inside a session should have a time-stamp belonging to a 30 minute interval.
Once that the sessions have been identified they have to be represented. Extending what has been done in [37] and [15] the considered features are described in table 6.2.

One possible additional feature can be the session IP membership to a DNS Black List (DNSBL). This information can be very meaningful as it can tell us if the IP belongs to one of the following categories:

- Tor;
- Proxy;
- Spammer;
- Zombie;
- Dial-up.

However finding this information for each session can be time expensive, even using an ip cache because to do so the ip has to be searched in very long lists of ips.

Name	Description
Total hits	Total number of HTTP request
% img	Percentage of images requested
% HTML	Percentafe of html file requested
% Binary doc	Percentage of binary doc files
% Binary exe	Percentage of binary executable files
% ASCII	Percentage of ASCII files
% Zip	Percentage of compressed files
% Multimedia	Percentage of multimedia files
% Other files	Percentage of other file requested
Bandwidth	Total bytes requested
Robots.txt	True if robot.txt has been visited
Session time	time passed between the first and the last connection
Avg interval	Avg time passed between two request
Is interval constant	True if the time between all couples of two following connections in the session is constant
Night requests	Number of requests between 12 p.m and 7 a.m.
Repeated requests	Number of repeated requests
% Errors	Percentage of requests resulting with code >= 400
% GET	Percentage of GETs
% POST	Percentage of POSTs
% HEAD	Percentage of HEADs
% Other Method	Percentage of other methods
% Unassigned Referrer	Percentage of requests with unassigned referrer
nMisbehavior	The number of requests signaled by the heuristics used to pre-filter the connections
nAloneMisbhavior	The number of requests with a fingerprint which signals an attack even alone
nOtherMisbehavior	The number of requests with a fingerprint which signals an attack only if related to other ones
geoIp	The origin of the request
dnsbl	True if the src ip is present in some DNS black list

Table 6.2: Session features. The ones with yellow background are optionals and can be added or not depending on the kind of model is going to be made.

6.3 Proposed Approach

As we already said the idea is to apply outlier detection, but we have still not discussed how to do it. Having this in mind, a solution to the risky assumption of starting from mostly normal connection has been proposed. Moreover, related to single connections, we already considered to split the data by resource. In such a way we will have more similar substets of the original data. This step may give us some problems if a resource has a very small amount of connections, or if someone asks for a never seen resource. In the first case we can simply skip this step, and using the entire dataset. It will require more time, as the number of features will grow, but there is always the possibility to apply dimensionality reduction algorithms, as PCA, to manage this problem. In the second case we can simply mark a request to a never seen resource as an outlier, as it is requesting something that completely differs from what we had analyzed in the training phase.
In the following paragraph an outlier detection method for evaluating single connections will be proposed, as an extension of what has already been said.

Outlier Detection for single connections: Let's consider in this phase a the set of connections related to the most used resource. After preprocessing the data as described in 6.2.1, KMeans (with euclidean distance) is applied to find clusters inside the data. As Kmeans need numerical values the strings present in the dataset are converted in integer using an hash functions. An alternative to this approach is to use One Hot Encoding [2], which converts the problem of strings to a binary one adding one feature (column) for each possible couple feature-value in the original dataset and than using 1 or 0 to indicate the presence or absence of that co-occurence. In this work it has now been used because it cause a huge growth in the number of features.
Remember that this first training phase has to be done on the connections that have been marked as normal (non malign) by the heuristics. Kmeans

[2] `https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/` accessed: November 2018

requires the number of cluster as parameter, so the best value for this parameter needs to be found. In our example it appeared to be 4 as Kmeans project 4 clear clusters in the space (figure 6.3). The image shows that the connection marked as attacks by the heuristics appear quite far from the cluster centers, moreover it shows that near the big clusters we can find some outliers (isolated point). Another important aspect is that three of the four cluster contain only bot connection, while the fourth one contains all the human connections, plus some other bot connection. This means that using clustering we can solve another of the big problems of the data under analysis, the fact the the most of the connections are coming from bots. This is a problem because it can lead to a model describing only bot connections, making result as outliers all the connections coming by humans. Clustering in some sense is solving the problem as it split the connections between group of bot ones, and human ones. Each group is then representing similar connections, so similar pattern, that can be used separately for outlier detection. Moreover using clustering to split the connections related to a single resource can be useful as it creates groups of similar items, improving like this the precision of the normal patterns generated. Starting from this point there are two alternatives to detect outliers:

- use a distance based threshold for each one of the clusters;
- make a one-class svm classifier for each one of the clusters.

The first method can be easier, because it does not need any additional computation or model training. The items distances from the cluster centers are already computed when the Kmeans model is fitted. This distances will be compared against a threshold and whenever an item distance is greater then the threshold it will be considered to be an outlier. The only difficulty is to understand how to fix the threshold over which consider an element to be an outlier. If we want to fix the number of false positives under a predefined percentage, n%, we can order the distances and take as threshold the distance value that is bigger than the n% of the distances. As an alternative

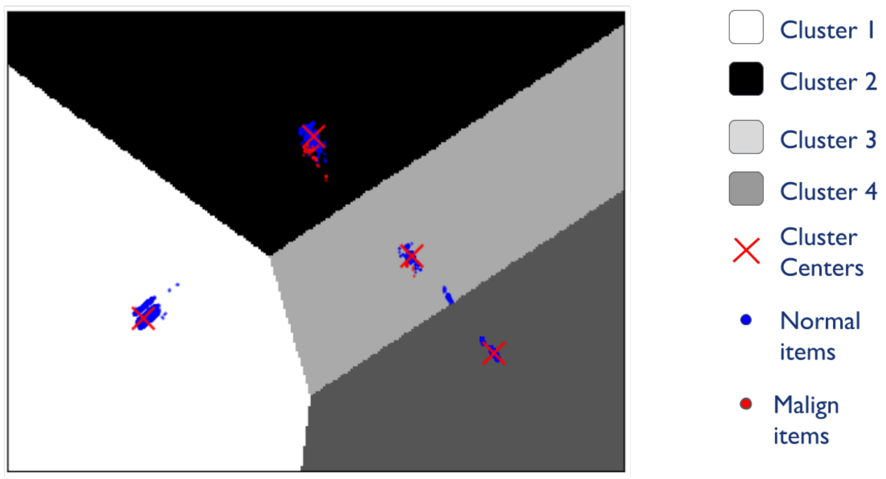


Figure 6.3: Kmeans Clustering output, projected in a 2 dimensional space using Principal Component Analysis. The blue and the red points represent respectively the normal and the attacks, as marked by the heuristics.

we can try other values, for example coming from the average distance and the mean deviation. In any case to improve the detection performance the threshold needs to by adjusted, but without having labelled data it is not possible.

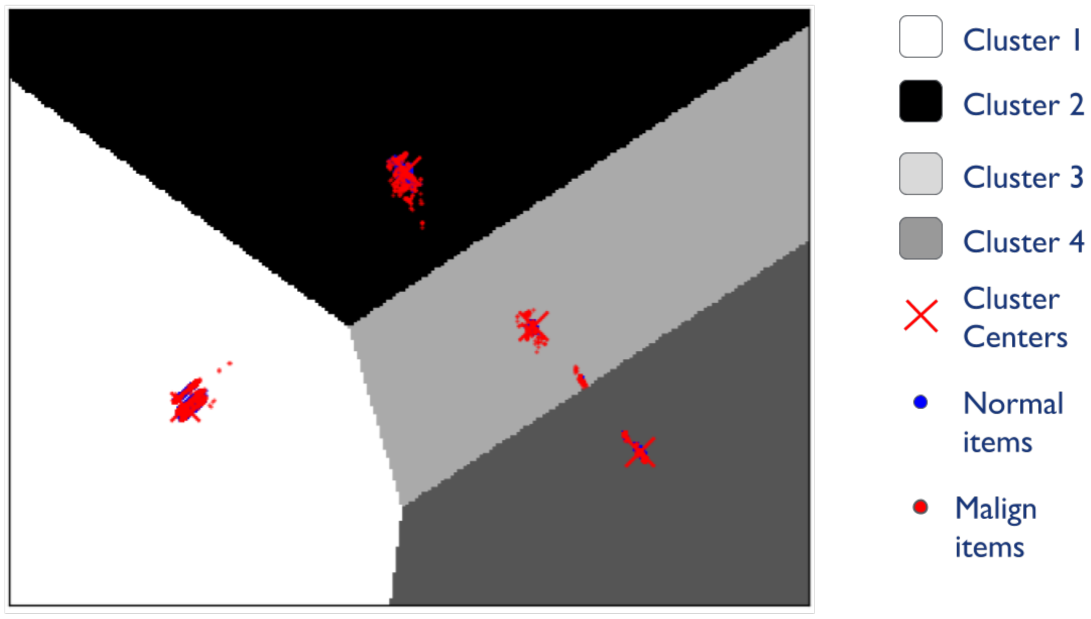
The second method is more expensive in terms of computation time, as it requires to train other models, in particular one One-Class SVM for each cluster. By the other side, as seen in the state of the art, this kind of classifier can be very precise detecting outliers and it finds them not simply fixing a distance threshold. Unluckily also in this case the model requires some parameter to be calibrated. The first is ν, which in some sense regulates the number of elements that can be considered to be outliers in the training data, so in our case it actually fix an upper bound to the percentage of outliers we will find in the data. While the second one if γ which is the coefficient of the svm kernel function. Unluckily also in this case these parameters need to be calibrated to improve the performances, but again, without labelled data this cannot be done.

Without having the possibility to evaluate the output of the models trying to find better parameters, what has been done is to fix a quite conservative thresholds; meaning thresholds that try to keep the number of detected outliers low, and so, also the number of false alarms. This choice has been done both for the distance based and for the One-Class SVM based outlier detection models, setting the distance threshold to 1.0 (around the double of the average distance to the cluster centers) and ν to 0.01 respectively. As first "evaluation" the percentage of connection marked as outliers that were also marked as attack by the heuristics has been counted. Moreover the number of outliers found in the connections marked as normal by the heuristics has been counted. These preliminary results are reported in table 6.3 which shows that the most of the connections marked as attacks by the heuristics are also marked as outliers by the models. This means that both the models are at able to detect the attacks found by the heuristics. Moreover we should consider that also the heuristics may have some false positive, and that we can change the models threshold to increase the percentages reported in table. The problem again is that without the labelled data we cannot know if there are false positives in the heuristics which are not detected by the models (so it is working fine) or if we need to improve its detection performances.

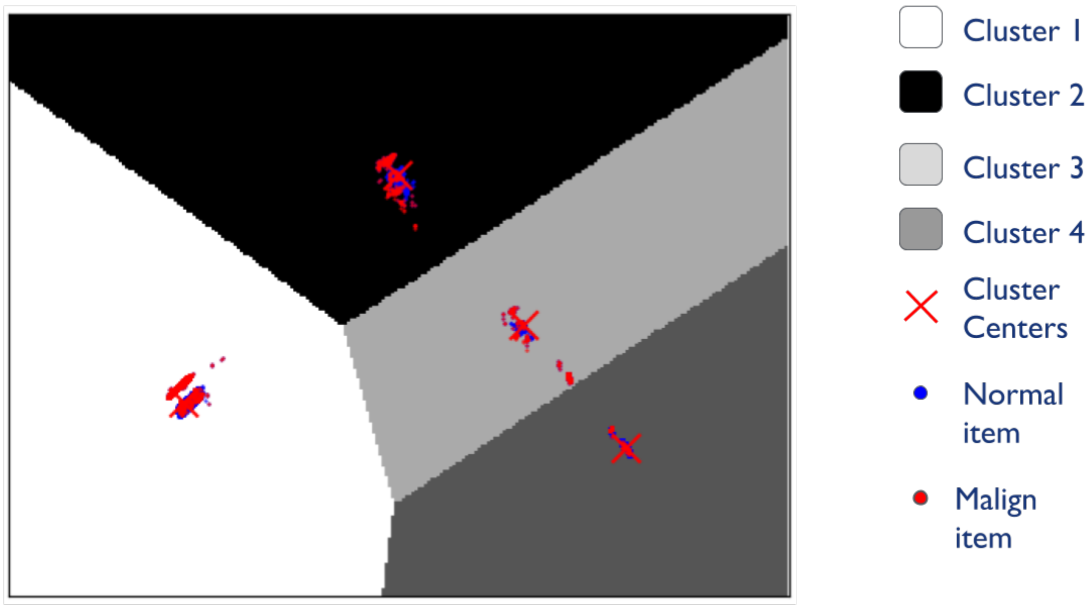
The same results are graphically expressed in figure 6.4, where you can see the other outliers found by the models. These will look the majority of the connections, but this is only a graphical issue due to the overlapping of the point. To understand the cardinality of the outliers refer to the previous table.

6.4 Preliminary Evaluation

Has already mentioned the data used for this experiment is completely unlabelled, thus not allowing an evaluation to be carried out. For this reason a domain expert have been asked to label a small subset of the original



(a) Distance based model.



(b) One-Class SVM based model.

Figure 6.4: Output of the models based on Kmeans + distance based or One-Class SVM based outlier detection.

	Known attacks marked as outliers	Outliers in training data	Outliers in cluster 0	Outliers in cluster 1	Outliers in cluster 2	Outliers in cluster 3
Distance based	85 %	22313 over 422006	21364 over 288478	230 over 25171	221 over 78426	498 over 29931
One-Class SVM based	90 %	4218 over 422006	2886 over 288478	250 over 25171	783 over 78426	299 over 29931

Table 6.3: Preliminary results of outlier detection on single connections. With known attacks we mean the connections marked as attacks by the heuristics, while the training data consists in the connection marked as non-attacks by the heuristics.

data. The Evaluation subset includes the 1% of the original dataset, obtained selecting the 1% of the connections related each one of the resources thus reproducing the original connection distribution. Moreover, related to the three most used resources the half of the connections included in the subset have been marked as outliers by the proposed models. In addition to this 1% of the Dataset (7262 items) a small dataset containing only attacks has been added (136 additional connections). The resulting evaluation dataset is unbalanced, as it contains only 325 attacks; 189 coming from the 1% of the original dataset plus other 136 coming from the additional connections (fig 6.5).
Moreover their threshold have been changed, to see how their performances change.

This evaluation process have been repeated two times, one considering only the most requested resource (which covers more than the 60% of all the connections) and one considering all the resources, without splitting by resource.

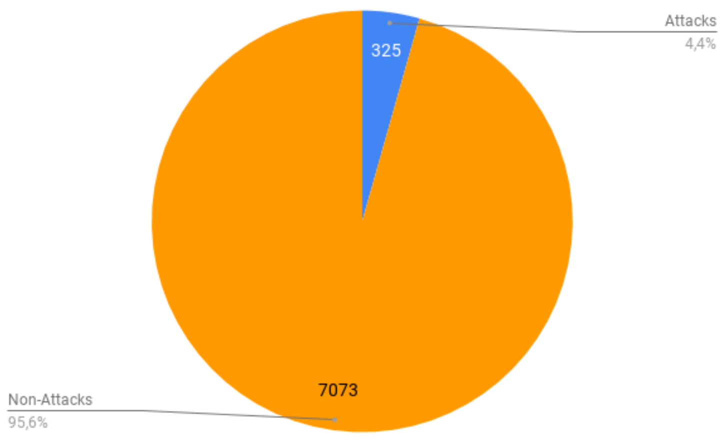


Figure 6.5: Evaluation dataset composition. Over a dataset containing 7398 connections only the 4.4% are attacks (325).

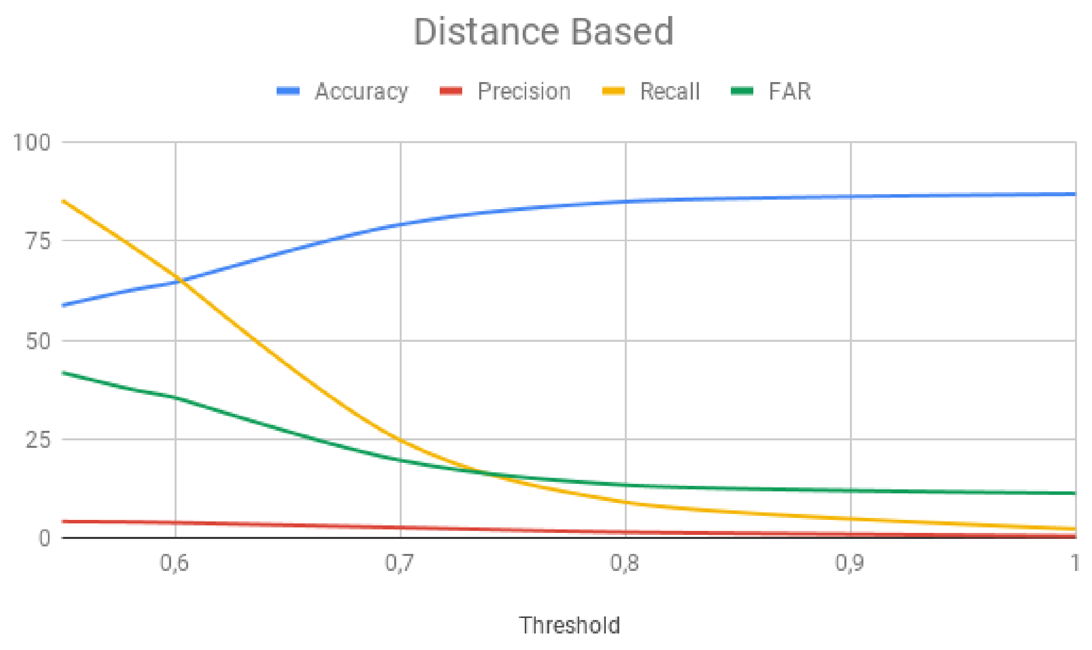
6.4.1 Evaluation on the most asked resource

The first evaluation has been done following the idea of splitting the connections by resource, focusing on the most used one. Doing this a problem emerged: in the connection marked by the expert there are no attacks to this resource. While the additional log file containing only attacks contains 89 attacks requesting this resource. Having these problems in mind, in figure 6.6 and table 6.4 you can see the results of both the models. The figure represent how the models behave while their parameters change, while the table shows which one goes better according to each one of the evaluation metrics. As opposed to what has been observed in the previous chapter we can observe that the distance based model can detect more attacks. Moreover his false alarms rate is lower as compared to the One-Class SVM based model. In both cases the precision is very low, but we have to consider that it is given by $\frac{TP}{TP+FP}$. Considering that we are applying an anomaly detection method the false positive will always be in some sense high, while we have a really low number of attacks, so also the true positives won't ever be enough to make up for the false positives. Although the results obtained from the proposed models are not optimal, it is possible to observe that they are still better than those obtained from heuristics:

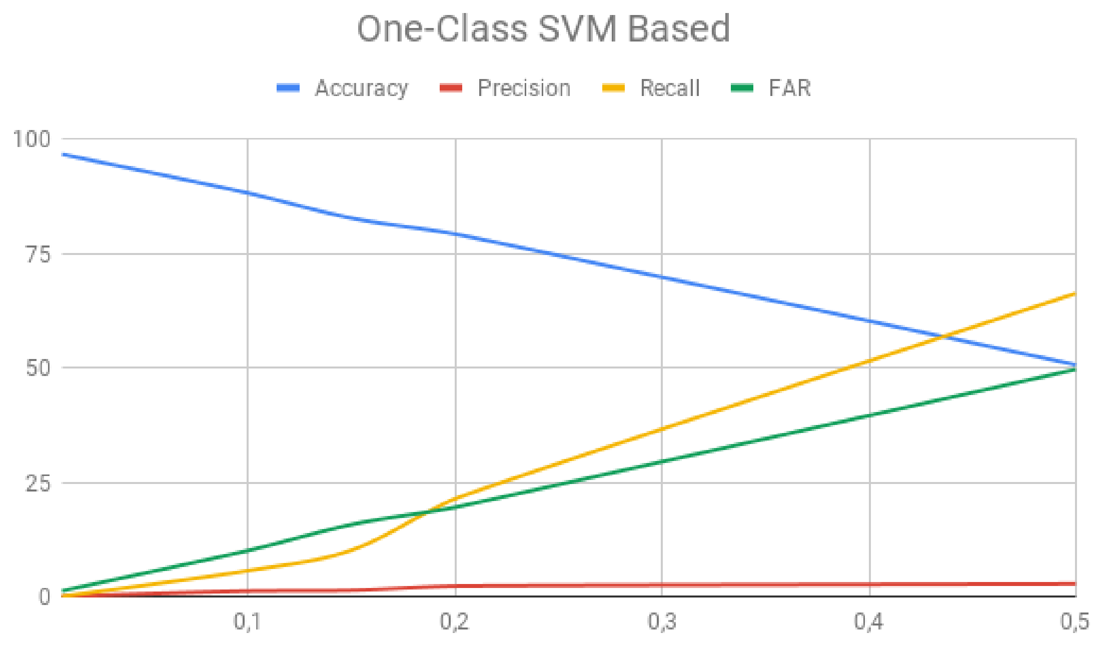
- Accuracy 97,93%
- Precision -
- Recall 0
- FAR 0

The reason why the accuracy is so high is that the heuristics are marking all the new connections as *non-attacks* including the 89 new ones, but in respect to the amount of connections this does not count a lot. The new attacks are not detected by the heuristics because they do not contain anyone of the most know attack fingerprints, and so they can pass undetected. Of course, though not clearly, these connections are different from the "normal" ones, and for this reason they can be detected by the outlier detection models if they are enough sensitive. But increasing their sensitivity also increase the number of false positives. In table 6.5 the performances of the heuristics and the best outlier detection model are compared with random guessing, which is often used as simple baseline.

As previously said the dataset used for the evaluation is strongly unbalanced, causing many problems. One example is the Precision, which always result very low. To solve this problem the evaluation has been repeated on a more balanced dataset, downsampling the number of normal (non-attack) connections to the double of the attacks. Table 6.6 shows the result of this second evaluation, from which is possible to observe how the Distance-Based outlier detection outperforms the One-Class SVM based. Moreover it is possible to observe how using a balanced dataset the general performances increase and in particular the precision, which was low using the unbalanced one. The same results are also reported in a graphical way in figure 6.7.



(a) Distance based model.



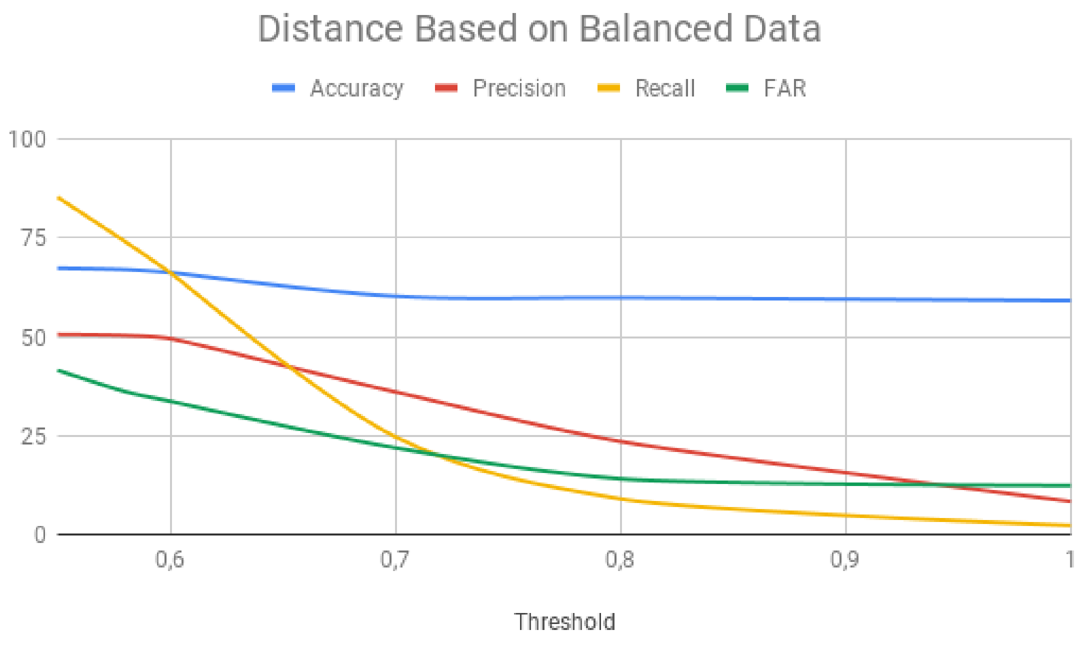
(b) One-Class SVM based model.

Figure 6.6: Proposed model performances on the most used resource.

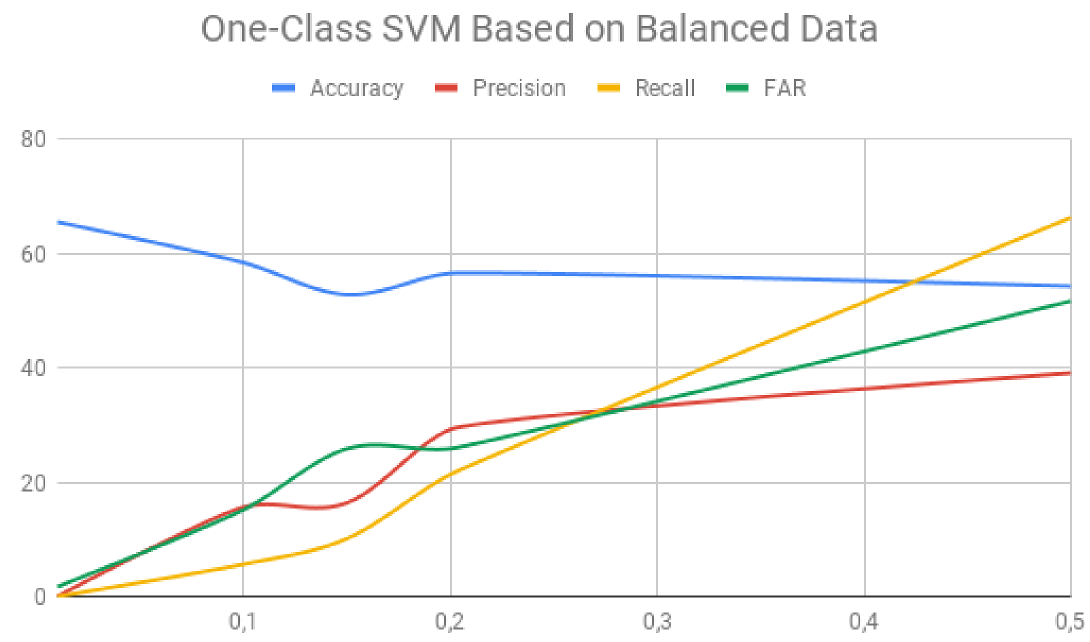
	Accuracy	Precision	Recall	FAR
Distance Based - *th* 0.55	58.77	4.12	85.39	41.81
Distance Based - *th* 0.58	62.57	3.98	74.15	37.66
Distance Based - *th* 0.6	64.59	3.79	66.29	35.44
Distance Based - *th* 0.7	79.23	2.58	24.72	19.61
Distance Based - *th* 0.8	85.05	1.4	8.98	13.33
Distance Based - *th* 1.0	86.98	0.42	2.24	11.23
One-Class SVM based- ν 0.01	96.75	0	0	1.2
One-Class SVM based- ν 0.1	88.23	1.16	5.61	10.02
One-Class SVM based- ν 0.15	82.76	1.33	10.11	15.7
One-Class SVM based- ν 0.2	79.28	2.25	21.34	19.49
One-Class SVM based- ν 0.5	50.67	2.73	66.29	49.65

⬜	Worst in column
🟧	Best in column

Table 6.4: Outlier detection results over the most used resource. For every evaluation metric the best and the worst models have been highlighted.



(a) Distance based model.



(b) One-Class SVM based model.

Figure 6.7: Proposed model performances on the most used resource when using a balanced Dataset.

	Accuracy	Precision	Recall	FAR
Distance Based th 0.5	58.77	4.12	85.39	41.81
Heuristics	97.93	-	0	0
Random guessing labels	51.74	2.29	53.93	48.30

Table 6.5: Evaluation over the most used resource and using the unbalanced dataset. Comparison of the best performing outlier detection model (in terms of recall), with the heuristics and random guessing labels.

6.4.2 Evaluation on the whole dataset

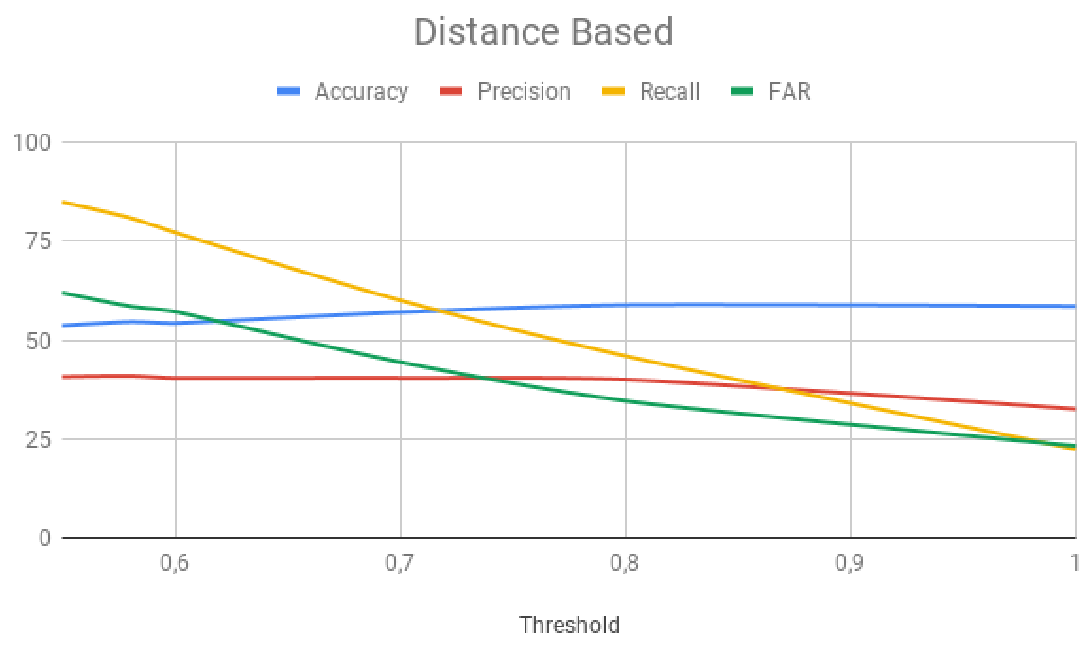
This time the proposed outlier detection models have been evaluated without considering the resources separately, but all together. The reason for this additional evaluation is that, as previously said, it may happen that some resource has not enough connection to build a model. As the previous section shown how using an unbalanced dataset for the evaluation creates some problem this time only the balanced one has been used. As previously done, the balanced evaluation dataset has been obtained downsampling the non-attack connection to the double of the attack ones.
The results of this evaluation are described in table 6.7 and figure 6.8. In the table also the performances of the heuristics and random guessing labels are expressed as a baseline. As expected the heuristics show low false positives, but also a low recall (detection ratio). The reason for this behavior is that heuristics belong to the misuse intrusion detection family, which is not able to detect attacks that have not been previously described.
Moreover we can observe how the performances are in general worst than the one obtained splitting by resource. Also this behavior was expected because splitting by resource we obtain clusters of more homogeneous connections compared to the one obtained here. This cause poorer normal pattern representations and so worst detection performances. This phenomenon highly

	Accuracy	Precision	Recall	FAR
Distance Based - th 0.55	67.41	50.66	85.39	41.57
Distance Based - th 0.58	67.04	50.38	74.15	36.15
Distance Based - th 0.6	66.29	49.57	66.29	33.70
Distance Based - th 0.7	60.29	36.06	24.71	21.91
Distance Based - th 0.8	59.92	23.52	8.98	14.06
Distance Based - th 1.0	59.17	8.33	2.24	12.35
One-Class SVM Based - ν 0.01	65.54	0.0	0.0	1.68
One-Class SVM Based - ν 0.1	58.42	15.62	5.61	15.16
One-Class SVM Based - ν 0.15	52.80	16.36	10.11	25.84
One-Class SVM Based - ν 0.2	56.55	29.23	21.34	25.84
One-Class SVM Based - ν 0.5	54.30	39.07	66.29	51.68

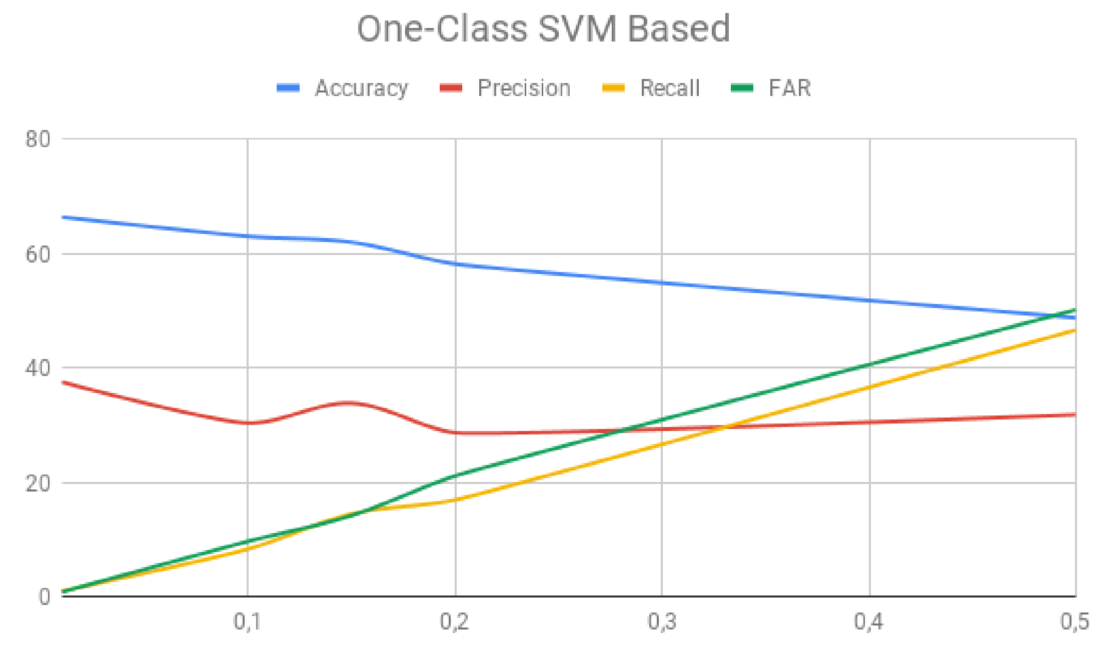
⬜	Worst in column
🟧	Best in Column

Table 6.6: Outlier detection results over the most used resource using a balanced dataset. For every evaluation metric the best and the worst models have been highlighted

affects the One-Class SVM Based model, whose performances diminishes to such an extent that the FAR is superior to the recall. Lastly this experiment confirms that the model based on distance works better than the One-Class SVM based. But differently from the previous evaluations the best threshold is not 0.55, as it lead to an excessive number of false alarms (FAR = 62%). In this case probably the best performing model can be considered the Distance-based one with threshold = 0.7 as it leads to an accuracy of 57.06% ,a precision of 40.41% , a recall of 60.12% and a FAR of 44.46%.



(a) Distance based model.



(b) One-Class SVM based model.

Figure 6.8: Proposed model performances without splitting the connections by resource.

	Accuracy	Precision	Recall	FAR
Distance Based - th 0.55	53.68	40.73	84.96	62.0
Distance Based - th 0.58	54.61	40.93	80.98	58.61
Distance Based - th 0.6	54.30	40.38	77.30	57.23
Distance Based - th 0.7	57.06	40.41	60.12	44.46
Distance Based - th 0.8	58.91	40.0	46.01	34.61
Distance Based - th 1.0	58.60	32.58	22.39	23.23
One-Class SVM Based - ν 0.01	66.39	37.5	0.92	0.76
One-Class SVM Based - ν 0.1	63.01	30.33	8.28	9.53
One-Class SVM Based - ν 0.15	61.98	33.81	14.41	14.15
One-Class SVM Based - ν 0.2	58.19	28.64	16.87	21.07
One-Class SVM Based - ν 0.5	48.77	31.79	46.62	50.15
Heuristics	93.78	22.76	17.17	2.68
Random guessing labels	49.69	33.26	50.30	50.61

⬜	Worst in Column
🟧	Best in Column

Table 6.7: Outlier detection results without splitting by resource and using a balanced dataset. Also the heuristics and random guessing performances are expressed as baseline. For every evaluation metric the best and the worst models have been highlighted.

Chapter 7

Conclusion and Future Works

This work firstly analyzed the state of the art about two important fields of cyber security: vulnerability and intrusion detection, focusing on the second one. Two important problems in this field emerged:

- The most of the proposed approaches are supervised, while in real world scenarios we rarely come across the labeled data.
- The state of the art methods are not reproducible as well as the datasets used for testing these approaches along with their source code are not available freely.

Moreover three Intrusion Detection System approaches based on Machine Learning have been reproduced on a common environment showing how Hybrid and Anomaly detection can be effective detecting known and new attacks. In detail the idea of applying One-Class SVM classifier on subsets of the data containing similar items emerged has the most powerful method.
Lastly a real world scenario has been considered. Differently from the most of the methods analyzed in the state of the art the data sources consisted of raw web server logs without any labels. The proposed solution takes into account both the most important aspects of Machine Learning: the data preprocessing and the ML method itself. As from the state of the art outlier detection emerged, the idea has been to apply this technique also in this

scenario, but it requires the assumption to start from a majority of *normal* (non attacks) data. To make this assumption more reliable a set of heuristics has been defined with the purpose of pre-filtering the raw logs. Moreover two feature extraction methods have been proposed, one to describe single connections and one to describe sessions. The reason for this choice is that these two different points of view can detect different kind of attacks. Lastly an outlier detection model has been proposed to analyze the single connections, based on the partitioning of the data by the requested resource and clustering, followed by outliers identification basing on distance metrics or on One-Class SVM models. In future also techniques that can be applied to the entire dataset needs to be considered. Moreover also some technique to automatically find the number of clusters should be determined, like the elbow method [1] which looks at the percentage of variance expressed by the data as a function of the number of clusters.

As the used data is completely unlabeled it has not been possible to make a precise evaluation of the methods, but it has been observed that both the outlier detection models (the distance based and the One-Class SVM based) are able to mark as outliers an high percentage of the connections marked as attack by the heuristics. This means that the proposed approaches are well reproducing the detections capability of the heuristics, also considering that in some case the heuristics themselves may give false positives. Moreover, A preliminary evaluation has been conducted where an expert labeled a small subset ($\sim 1\%$) of the original data. The subset was extracted based on the frequently requested resources. The evaluation was also conducted on the whole data set leading to two main result:

- The distance based model outperforms the One-Class SVM based and also the heuristics, showing how the outlier detection can be used to detect new attacks, even if the number of false alarms tends to grow.

- When the resources are considered one at a time, the performances are

[1] `https://en.wikipedia.org/wiki/Elbow_method_(clustering)` accessed: November 2018

significantly better than when they are all processed together. This demonstrates that defining techniques to group the data in similar clusters improves the outlier detection performances.

As a future perspective, similar methods should be tested on the dataset describing sessions. However, a different family of algorithms can be employed based on the requirements of the sessions. Precise evaluation strategies need to be designed in future on bigger and more balanced evaluation dataset. Lastly there is a need to define a model that analyzes application logs and generates some results. As a next step, these results should be combined with the ones coming from the session and the connection analysis.

Bibliography

[1] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection." In: (2018).

[2] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. "Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey." In: *ACM Comput. Surv.* 50.4 (Aug. 2017), 56:1–56:36. DOI: `10.1145/3092566`.

[3] Yisroel Mirsky, Tal Halpern, Rishabh Upadhyay, Sivan Toledo, and Yuval Elovici. "Enhanced situation space mining for data streams." en. In: *Proceedings of the Symposium on Applied Computing, SAC 2017, Marrakech, Morocco, April 3-7, 2017.* Ed. by Ahmed Seffah, Birgit Penzenstadler, Carina Alves, and Xin Peng. ACM Press, 2017, pp. 842–849. DOI: `10.1145/3019612.3019671`.

[4] Abdulmohsen Almalawi, Adil Fahad, Zahir Tari, Abdullah Alamri, Rayed AlGhamdi, and Albert Y. Zomaya. "An Efficient Data-Driven Clustering Technique to Detect Attacks in SCADA Systems." In: *IEEE Transactions on Information Forensics and Security* 11.5 (May 2016), pp. 893–906. DOI: `10.1109/TIFS.2015.2512522`.

[5] Elisa Bertino, Ravi Sandhu, and Alexander Pretschner, eds. *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy, CODASPY 2016, New Orleans, LA, USA, March 9-11, 2016.* ACM, 2016.

[6] Anna L. Buczak and Erhan Guven. “A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection.” In: *IEEE Communications Surveys Tutorials* 18.2 (2016), pp. 1153–1176. DOI: 10.1109/COMST.2015.2494502.

[7] Gustavo Grieco, Guillermo Luis Grinblat, Lucas C. Uzal, Sanjay Rawat, Josselin Feist, and Laurent Mounier. “Toward Large-Scale Vulnerability Discovery using Machine Learning.” In: *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy, CODASPY 2016, New Orleans, LA, USA, March 9-11, 2016*. Ed. by Elisa Bertino, Ravi Sandhu, and Alexander Pretschner. ACM, 2016, pp. 85–96. DOI: 10.1145/2857705.2857720.

[8] Jihyun Kim, Jaehyun Kim, Huong Le Thi Thu, and Howon Kim. “Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection.” In: *2016 International Conference on Platform Technology and Service (PlatCon)*. Feb. 2016, pp. 1–5. DOI: 10.1109/PlatCon.2016.7456805.

[9] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. “The FAIR Guiding Principles for scientific data management and stewardship.” In: *Scientific data* 3 (2016).

[10] Awad A. Younis, Yashwant K. Malaiya, Charles Anderson, and Indrajit Ray. “To Fear or Not to Fear That is the Question: Code Characteristics of a Vulnerable Functionwith an Existing Exploit.” In: *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy, CODASPY 2016, New Orleans, LA, USA, March 9-11, 2016*. Ed. by Elisa Bertino, Ravi Sandhu, and Alexander Pretschner. ACM, 2016, pp. 97–104. DOI: 10.1145/2857705.2857750.

[11] Salma Elhag, Alberto Fernández, Abdullah Bawakid, Saleh Alshomrani, and Francisco Herrera. “On the combination of genetic fuzzy

systems and pairwise learning for improving detection rates on Intrusion Detection Systems." en. In: *Expert Systems with Applications* 42.1 (Jan. 2015), pp. 193–202. DOI: `10.1016/j.eswa.2014.08.002`.

[12] Wei-Chao Lin, Shih-Wen Ke, and Chih-Fong Tsai. "CANN: An intrusion detection system based on combining cluster centers and nearest neighbors." en. In: *Knowledge-Based Systems* 78 (Apr. 2015), pp. 13–21. DOI: `10.1016/j.knosys.2015.01.009`.

[13] Henning Perl, Sergej Dechand, Matthew Smith, Daniel Arp, Fabian Yamaguchi, Konrad Rieck, Sascha Fahl, and Yasemin Acar. "VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits." In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015.* Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. ACM, 2015, pp. 426–437. DOI: `10.1145/2810103.2813604`.

[14] Douglas Reynolds. "Gaussian mixture models." In: *Encyclopedia of biometrics* (2015), pp. 827–832.

[15] Dilip Singh Sisodia, Shrish Verma, and Om Prakash Vyas. "Agglomerative Approach for Identification and Elimination of Web Robots from Web Server Logs to Extract Knowledge about Actual Visitors." en. In: *Journal of Data Analysis and Information Processing* 03.01 (2015), pp. 1–10. DOI: `10.4236/jdaip.2015.31001`.

[16] Fabian Yamaguchi, Alwin Maier, Hugo Gascon, and Konrad Rieck. "Automatic Inference of Search Patterns for Taint-Style Vulnerabilities." In: *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society, May 2015, pp. 797–812. DOI: `10.1109/SP.2015.54`.

[17] Leyla Bilge, Sevil Sen, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. "Exposure: A Passive DNS Analysis Service to Detect and Report Malicious Domains." en. In: *ACM Transactions on Information and System Security* 16.4 (Apr. 2014), pp. 1–28. DOI: `10.1145/2584679`.

[18] Gisung Kim, Seungmin Lee, and Sehun Kim. “A novel hybrid intrusion detection method integrating anomaly detection with misuse detection.” In: *Expert Systems with Applications* 41.4 (2014), pp. 1690–1700. DOI: `10.1016/j.eswa.2013.08.066`.

[19] Riccardo Scandariato, James Walden, Aram Hovsepyan, and Wouter Joosen. “Predicting vulnerable software components via text mining.” In: *IEEE Transactions on Software Engineering* 40.10 (2014), pp. 993–1006. DOI: `10.1109/TSE.2014.2340398`.

[20] Aram Hovsepyan, Riccardo Scandariato, Wouter Joosen, and James Walden. “Software vulnerability prediction using text analysis techniques.” In: *Proceedings of the 4th international workshop on Security measurements and metrics*. ACM. 2012, pp. 7–10.

[21] Jens Müller, Jörg Schwenk, and Ing Mario Heiderich. “Web Application Forensics.” In: (2012).

[22] Hossain Shahriar and Mohammad Zulkernine. “Mitigating program security vulnerabilities: Approaches and challenges.” In: *ACM Computing Surveys (CSUR)* 44.3 (2012), p. 11. DOI: `10.1145/2187671.2187673`.

[23] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. “EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis.” In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*. The Internet Society, 2011.

[24] Istehad Chowdhury and Mohammad Zulkernine. “Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities.” In: *Journal of Systems Architecture* 57.3 (2011), pp. 294–313. DOI: `10.1016/j.sysarc.2010.06.003`.

[25] Cynthia Wagner, Jérôme François, Radu State, and Thomas Engel. “Machine Learning Approach for IP-Flow Record Anomaly Detection.” en. In: *NETWORKING 2011*. Ed. by David Hutchison et al. Vol. 6640.

Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 28–39. DOI: 10.1007/978-3-642-20757-0_3.

[26] Mehran Bozorgi, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. "Beyond heuristics: learning to classify vulnerabilities and predict exploits." en. In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*. Ed. by Bharat Rao, Balaji Krishnapuram, Andrew Tomkins, and Qiang Yang. ACM Press, 2010, p. 105. DOI: 10.1145/1835804.1835821.

[27] Yu-Xin Ding, Min Xiao, and Ai-Wu Liu. "Research and implementation on snort-based hybrid intrusion detection system." In: *2009 International Conference on Machine Learning and Cybernetics*. Vol. 3. July 2009, pp. 1414–1418. DOI: 10.1109/ICMLC.2009.5212282.

[28] Ranjit Jhala and Rupak Majumdar. "Software model checking." In: *ACM Comput. Surv.* 41.4 (2009), 21:1–21:54. DOI: 10.1145/1592434.1592438.

[29] Adetokunbo A.O. Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. "Clustering event logs using iterative partitioning." en. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*. Ed. by John F. Elder IV, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki. ACM Press, 2009, p. 1255. DOI: 10.1145/1557019.1557154.

[30] Arman Tajbakhsh, Mohammad Rahmati, and Abdolreza Mirzaei. "Intrusion detection using fuzzy association rules." In: *Applied Soft Computing* 9.2 (Mar. 2009), pp. 462–469. DOI: 10.1016/j.asoc.2008.06.001.

[31] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. "A detailed analysis of the KDD CUP 99 data set." In: *2009 IEEE Symposium on Computational Intelligence for Security and Defense*

Applications, CISDA 2009, Ottawa, Canada, July 8-10, 2009. IEEE, 2009, pp. 1–6. DOI: 10.1109/CISDA.2009.5356528.

[32] Jiong Zhang, Mohammad Zulkernine, and Anwar Haque. "Random-Forests-Based Network Intrusion Detection Systems." In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.5 (Sept. 2008), pp. 649–659. DOI: 10.1109/TSMCC.2008.923876.

[33] Mark Dowd, John McDonald, and Justin Schuh. *The art of software security assessment: Identifying and preventing software vulnerabilities.* Pearson Education, 2006.

[34] Zhang Xue-qin, Gu Chun-hua, and Lin Jia-jun. "Intrusion detection system based on feature selection and support vector machine." In: *Communications and Networking in China, 2006. ChinaCom'06. First International Conference on.* IEEE. 2006, pp. 1–5.

[35] Christopher Kruegel, Giovanni Vigna, and William Robertson. "A multi-model approach to the detection of web-based attacks." en. In: *Computer Networks* 48.5 (Aug. 2005), pp. 717–738. DOI: 10.1016/j.comnet.2005.01.009.

[36] Wei Lu and Issa Traore. "Detecting New Forms of Network Intrusion Using Genetic Programming." en. In: *Computational Intelligence* 20.3 (Aug. 2004), pp. 475–494. DOI: 10.1111/j.0824-7935.2004.00247.x.

[37] Pang-Ning Tan and Vipin Kumar. "Discovery of Web Robot Sessions Based on Their Navigational Patterns." en. In: *Intelligent Technologies for Information Analysis.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 193–222. DOI: 10.1007/978-3-662-07952-2_9.

[38] Pingchuan Ma. "Log Analysis-Based Intrusion Detection via Unsupervised Learning." In: *Master of Science, School of Informatics, University of Edinburgh* (2003).

[39] Richard P. Lippmann and Robert K. Cunningham. "Improving intrusion detection performance using keyword selection and neural networks." en. In: *Computer Networks* 34.4 (Oct. 2000), pp. 597–603. DOI: 10.1016/S1389-1286(00)00140-7.

[40] Richard P. Lippmann, David J. Fried, Isaac Graf, Joshua W. Haines, Kristopher R. Kendall, David McClung, Dan Weber, Seth E. Webster, Dan Wyschogrod, Robert K. Cunningham, et al. "Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation." In: *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings.* Vol. 2. 2000, 12–26 vol.2. DOI: 10.1109/DISCEX.2000.821506.

[41] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. "The 1999 DARPA on-line intrusion detection evaluation." en. In: *Computer Networks* (2000), p. 17.

[42] Chan Man Kuok, Ada Fu, and Man Hon Wong. "Mining fuzzy association rules in databases." en. In: *ACM SIGMOD Record* 27.1 (Mar. 1998), pp. 41–46. DOI: 10.1145/273244.273257.

[43] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. "Mining Association Rules between Sets of Items in Large Databases." In: *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 26-28, 1993.* Ed. by Peter Buneman and Sushil Jajodia. Vol. 22. 2. ACM. ACM Press, 1993, pp. 207–216. DOI: 10.1145/170035.170072.

[44] Richard P. Lippmann, Linda Kukolich, and Elliot Singer. *LNKnet: Neural Network, Machine-Learning, and Statistical Software for Pattern Classification.* en. Tech. rep. MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB, Jan. 1993.

Sitography

[45] cgisecurity.com. *Fingerprinting Port 80 Attacks:A look into web server, and web application attack signatures.* `https://www.cgisecurity.com/papers/fingerprint-port80.txt`. (Visited on 10/24/2018).

[46] cgisecurity.com. *Fingerprinting Port 80 Attacks:A look into web server, and web application attack signatures: Part Two.* `https://www.cgisecurity.com/papers/fingerprinting-2.txt`. (Visited on 10/24/2018).

Acknowledgments

I would like to thank all those who helped me in the realization of this Thesis:
the supervisor Paolo Ciancarini;
the co-supervisor Valentina Presutti;
and the co-supervisor Mehwish Alam.