TÉCNICO LISBOA

STAKE: Secure Tracing of Anomalies using previous Knowledge and Extensions

Kevin Batista Corrales

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor(s): Prof. Miguel Filipe Leitão Pardal

Examination Committee

Chairperson: Prof. Daniel Jorge Viegas Gonçalves
Supervisor: Prof. Miguel Filipe Leitão Pardal
Member of the Committee: Prof. Ibéria Vitória de Sousa Medeiros

March 2021

Acknowledgments

I would like to thank my supervisor Prof. Miguel L. Pardal for providing continuous support, feedback and adjustment of this research.

I would also like to thank INESC-ID for providing equipment that allowed me to produce a real environment where the studied technologies were used.

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2019 (INESC-ID) and through project with reference PTDC/CCI-COM/31440/2017 (SureThing).

Finally, I would like to thank all my friends and family for encouraging me and supporting me during my research and studies.

Resumo

Os dispositivos de Internet das Coisas (*Internet of Things*, IoT) têm-se tornado mais presentes em nossas casas. No entanto, uma casa inteligente (*Smart Home*) é um ambiente desafiador. Este tipo de ambiente consiste em pessoas envolvidas com dispositivos para uma variedade de propósitos, do qual é difícil detectar anomalias que possam ser ciberataques. Nesta dissertação introduzimos o STAKE, uma solução para capturar e analisar o tráfego de rede numa *Smart Home*, com diferentes níveis de detalhes. O sistema suporta extensões (*plug-ins*) de detecção de anomalias para detectar ataques em tempo útil. Avaliamos a nossa proposta com *plug-ins* de *Machine Learning* baseados nos modelos *Elliptic Envelope* e *Random Forest*. STAKE foi capaz de executar diferentes *plug-ins* e ambos detectaram anomalias.

Conforme esperado, cada modelo retornou resultados diferentes. Ambos os *plug-ins* demonstraram resultados promissores quando foram criados. O modelo Elliptic Envelope obteve 93,9% de *accuracy* e o Random Forest obteve 96,7% de *accuracy*. No entanto, quando treinados novamente no sistema, os plug-ins não demonstraram ser flexíveis o suficiente para alterações na rede Smart Home. O plug-in Elliptic Envelope demonstrou tendência de produzir taxas de falso positivo excessivamente altas, o que deteriorou a diferença entre amostras benignas e anomalias nos dados de treinamento. O plug-in Random Forest demonstrou uma tendência excessiva para overfitting quando retreinado neste tipo de ambiente.

Palavras-chave: Internet das Coisas, Sistema de Detecção de Intrusões, Detecção de Anomalias, Aprendizagem Automática

Abstract

Internet of Things (IoT) devices have become more present in our households because of an increase in availability and affordability. However, a Smart Home is a challenging environment. It involves people engaged with devices for a variety of purposes, and it is difficult to detect anomalies that can be cyber attacks. In this dissertation we introduce STAKE, a Smart Home gateway for capturing and analysing network traffic, with different levels of detail. The system supports anomaly detection plug-ins to spot attacks in near real-time. We evaluated our system with Machine Learning plug-ins based on the Elliptic Envelope and the Random Forest models. STAKE was able to execute different plug-ins and both detected anomalies.

As expected, each model returned different results. Both plug-ins demonstrated promising results when they were created. The Elliptic Envelope model obtained 93,9% accuracy and the Random Forest obtained 96,7%. However, when re-trained in the system the plug-ins did not demonstrate being flexible enough to changes in the Smart Home network. The Elliptic Envelope plug-in demonstrated tendency to produce excessively high false positive rates, which deteriorated the difference between benign and anomaly samples in the training data. The Random Forest plug-in demonstrated a exceedingly tendency for overfitting when re-trained in this kind of environment.

Keywords: Internet of Things, Intrusion Detection System, Anomaly Detection, Machine Learning

Contents

List of Tables

List of Figures

Glossary

EE Elliptic Envelope consists in a robust co-variance estimate that assumes that the data is Gaussian distributed. It will define the shape of the data we have, creating a frontier that delimits the contour. 32

IDS Intrusion Detection System is a device or software application that monitors a network or systems for malicious activity or policy violations. 1

IoT Internet of Things is the network resulting from the connection of physical devices to the Internet and/or to each other. 1

RF Random Forest consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest splits out a class prediction and the class with the most votes becomes the model prediction. 34

RL Reinforcement learning is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. 15

SL Supervised Learning refers to a type of Machine Learning model category, where the algorithm is given labeled input data and the expected output results. 15

UL Unsupervised Learning refers to a type of Machine Learning model category, where the algorithm identifies patterns in the input data that are neither classified nor labeled. 15

Chapter 1

Introduction

The Internet of Things (IoT) consists of a network of billions (10^9) of interconnected devices, with embedded smart sensors and computational resources, which are connected to the Internet. IoT devices can connect and transfer data over a network without requiring direct human interaction [Hos18]. In many cases this makes the devices autonomous or semi-autonomous. However, IoT devices often have characteristics such as limited network connection, low processing power, low storage resources, reduced dimensions and low energy consumption.

The Smart Home is one of the promises of IoT. A Smart Home refers to a convenient home setup where devices can be remotely controlled, both manually or automatically [ZACF18]. The Smart Home is a challenging environment, because it involves people engaged with devices for a variety of purposes that involve real world interactions. Smart Home devices have become more affordable and available, and their presence in consumer households is increasing. The devices can be very diverse, ranging from simple light bulbs, smart plugs and locks, to more powerful devices, like video cameras, digital assistants, tablets and laptops. As a result, the amount of privacy-sensitive data uploaded to the cloud is also increasing. If proper security measures are not taken, these new Internet-connected devices can become entry points for an attacker.

Intrusion Detection System (IDS) is a common approach for network security. An IDS monitors the network and system activities, assesses the integrity of the system and data, recognizes malicious activity patterns, generates reactions to intrusions, and reports the outcome of detection [DD11]. An important method used by IDS is anomaly detection [HIZH19], which consists on finding patterns in data that do not conform to expected behaviour, for example, in Internet traffic [DAF18].

However, applying traditional IDS techniques to IoT is difficult due to the resource constraints of the devices and because the traffic pattern of smart devices is different from traditional network hosts [OM19]. The resources available on each IoT device are limited and the

volume of traffic generated by these devices is significant. Therefore, it is critical to choose an appropriate anomaly detection approach that can effectively and efficiently identify abnormal behaviours, to make real-time network level analysis possible and successful.

1.1 Objectives

In this work, we propose and implement a solution for capturing and storing Smart Home network traffic, and an execution environment for anomaly detection plug-ins. The network traffic is captured in different detail levels: traces, flows and summary features. The captured data is stored in a persistent repository with adequate schema and indexing. The plug-in execution environment allows using Machine Learning models, trained from the captured data, and applied to the near real-time detection of anomalies.

We called our solution STAKE, standing for: Secure Tracing of Anomalies using previous Knowledge and Extensions. It extends SPYKE [WPC19], a previous work that already performs device detection and applies knowledge-based rules, like quota limits per device. We implemented a Supervised Learning plug-in and an Unsupervised Learning plug-in.

1.2 Contributions

The fundamental contribution of this work is the development of STAKE. The system was designed, implemented[1] and tested as a Smart Home gateway device, standing between the user IoT devices and the cloud service providers. The STAKE anomaly detection plug-in engine provides a modular system that allows for swapping out, removing, and reimplementing Machine Learning algorithms. The plug-in engine allows both Unsupervised and Supervised learning approaches, as demonstrated by the Elliptic Envelope and Random Forest plug-ins, respectively.

There were a few more contributions from this work:

- A state-of-the-art summary on previous research of anomaly detection in Smart Home and IoT environments;
- A web-based management console implementation, allowing full configuration and operation of the system and its plug-ins, through an user-friendly interface;
- A dataset[2] with instance data collected from actual Smart Home devices in a realistic scenario, that can be reused for future works.

[1]The source code for STAKE is open-source and is available at: `https://github.com/inesc-id/STAKE`, accessed on March 29, 2021

[2]The dataset is publicly available at: `http://surething.tecnico.ulisboa.pt/files/STAKE-dataset.zip`, accessed on March 29, 2021

1.3 Thesis Outline

The remainder of the document is structured as follows: Chapter 2 presents the background surrounding the topic of our work and the related work focusing on anomaly detection and other security alternatives applied to IoT environments. Chapter 3 describes the proposed solution. Chapter 4 defines the evaluation approach. Chapter 5 describes the obtained results. Finally, Chapter 6 concludes this document.

Chapter 2

Background and Related Work

In this Chapter we present theoretical concepts to support our proposed solution, starting with Firewall and Intrusion Detection System network defences. Additionally, we look at Machine Learning (ML) and its use in Cybersecurity. At last, we present a literature review to support our proposed solution covering works on IDS, especially those proposed for IoT and the Smart Home.

2.1 Network Defence

Firewalls are perhaps the best-known defence in network systems. A Firewall is based on access control, enforcing predefined policies for how devices in the network are allowed to communicate with one another, filtering unauthorized traffic between devices in the network.

An *IDS* system monitors and analyses user behaviour and system activities, assesses the integrity of the system and data, recognizes malicious activity patterns, generates reactions to intrusions and reports the outcome of detection [CC18]. IDS detects attempts or successful breaches of a network by making passive observations of traffic. An IDS is composed of several modules: misuse/signature detection, anomaly detection algorithms, hybrid detection, profiling, privacy-preservation data mining, and scan detector [DD11].

The *misuse/signature detection* module matches malicious patterns with a high detection rate and a low false alarm rate. However, they cannot detect unknown attacks. The *anomaly detection algorithms* build normal patterns in a cyber-infrastructure, such that they can detect the patterns that deviate significantly from the normal model. They can detect new attacks. However, if normal data shows the same patterns as malicious data, the number of false alarms rises. The *hybrid detection* module is the aggregation from both mentioned modules above. It normally improves the detection rate and decreases the false alarm rate.

The *profiling* module performs clustering algorithms and/or other data mining to group similar network connections and search for dominant behaviours. *Privacy-preservation data mining* focuses in reducing unauthorized access of private information, while retaining the same functions as a normal data-mining method for discovering useful knowledge. The objective of this module is to prevent unauthorized users from accessing private information, such as private data mining or ML results. At last, the *scan detector* module finds vulnerabilities in cyber-infrastructures. Thus, this module is considered a preventive process.

The IDS system can be a Network-based Intrusion Detection System (NIDS) or a Host-based Intrusion Detection System (HIDS). NIDS monitors network traffic to identify different kinds of malicious activities. HIDS monitors the activity of a single host and the events occurring within that host. It generally monitors the log files, the various running processes and applications, file access and modification, system and application configuration changes. IDS have been an important tool for the protection of networks and information systems so it would be useful if they could also be applied to IoT deployments and use the latest advances in data processing, such as ML.

2.2 Machine Learning

Machine Learning (ML) is a set of mathematical techniques, implemented on computer systems, to perform information mining, pattern discovery, and inferencing from data [DD11].

2.2.1 Approaches and Components

ML can be divided into four approaches: Supervised Learning, Unsupervised Learning, Semi-supervised Learning, and Reinforcement Learning.

Supervised Learning adopts a knowledge discovery approach, using probabilities of previously observed events to infer the probabilities of new events. *Unsupervised Learning* methods draw abstractions from unlabeled datasets and apply these to new data. Both families of methods above can be applied to problems of classification (assigning observations to categories) or regression (predicting numerical properties of an observation). Additionally, there is *Semi-supervised Learning* [CC18]. In this hybrid type of learning, the model starts with training data that is labeled with the correct answers and then concludes with a model with a tuned set of weights, which is able to predict results for similar data that have not already been labeled. Finally, *Reinforcement Learning* can be used with both labeled and unlabeled data. Wang et al. [WZZ19] considers *Reinforcement Learning* the study of the problem to achieve the best trade-off between *exploitation* of current knowledge and *exploration* of a black box environment.

All ML algorithms are defined by three interdependent components [CC18]: model family, loss function and optimization procedure.

A *model family* shares a set of characteristics: computational complexity, mathematical complexity and explainability [CC18]. Computational complexity affects the training duration of the model. Using too much data available may affect the model negatively. Mathematical complexity also impacts the model training. Although, many datasets have non-linear boundaries, the data might be such that a linear decision boundary will provide good classification. Finally, explainability states how clear the model is able to explain why and how it achieved that decision or classification.

A *loss function* allows us to quantitatively compare different models. This function is used to measure the "cost" of wrong predictions or the "loss" associated with them. Mathematically, a loss function is a function that maps a set of pairs of predicted label and truth label to a real number. *Optimization* allows us to search for the optimal set of parameters that minimizes the loss function. This procedure is executed iteratively by comparing solutions until an optimal or a satisfactory solution is found. There are a diverse range of algorithms for optimization, including gradient-based algorithms, derivative-free algorithms and meta-heuristics [BCN18].

2.2.2 Evaluation Metrics

Even though Unsupervised Learning models do not use labels to train, when labels are available, they can be used to evaluate this type of models. Labeled data allows us to obtain True Positives, False Positives, True Negatives and False Negatives values that let us calculate the following equations.

In general, using only *accuracy* (as defined in equation 2.1) to measure model prediction performance is not enough, since it is abstract and only provides an approximate measure of a model performance. The accuracy is a simple way of measuring the effectiveness of a model, but it can be misleading. Therefore, other measures are necessary for us to be able to obtain a concrete understanding of the models behaviour and efficiency. Additionally, accuracy is sensitive to any change in the data set and is mostly effective when data are not balanced.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{TN} + \text{FP}} \ , \tag{2.1}$$

where TP = True Positive, FN = False Negative, TN = True Negative and FP = False Positive

Therefore, it was also considered other type of evaluation metrics. In an anomaly detection context, to comprehensively evaluate imbalanced learning, especially for minority classification, it is commonly used methods such as [DD11]: precision, recall, f-score and ROC curve (Receiver

Operating Characteristics) plots.

In short, *precision* (defined in equation 2.2) is the fraction of relevant instances among the retrieved instances. On the other hand, *recall* (equation 2.3) represents the fraction of the total amount of relevant instances that were actually retrieved. In other words, *precision* measures the degree to which repeated measurements under unchanged conditions show the same results and *recall* measures the proportion of positives that are correctly identified [KDL19].

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \ , \tag{2.2}$$

where TP = True Positive and FP = False Positive

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \ , \tag{2.3}$$

where TP = True Positive and FN = False Negative

Possessing these values, we are able to calculate *f-score* (equation 2.4), which allows us to measure the accuracy of the test. The f-score represents a weighted harmonic mean of the precision and recall of the test.

$$\text{F-score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.4}$$

Furthermore, the ROC curve is generated by plotting the True Positive Rate, also known as recall (equation 2.3), against the False Positive Rate (equation 2.5) at various threshold settings.

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} = 1 - \text{Specificity} \ , \tag{2.5}$$

where FP = False Positive and TN = True Negative and

$$\text{Specificity} = \frac{\text{TN}}{\text{N}} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{2.6}$$

The ROC measurement is a probability curve and AUC (Area Under The Curve) represents degree or measure of separability. AUC measurement represents how much model is capable of distinguishing between classes. The higher the AUC, the better the model is at predicting.

2.3 Smart Home Security Monitors

The baseline of this project is SPYKE [WPC19], an open source network intermediary for Smart Home networks, that provides communication monitoring between devices and remote servers, and also the ability to block and limit unwanted connections.

SPYKE's architecture is subdivided into two modules: the first one is authentication, where devices authenticate with the gateway in order to make requests; the second one is user policy enforcement that SPYKE uses for providing privacy protection. Figure 2.1 presents an overview of the entirety of the SPYKE system.

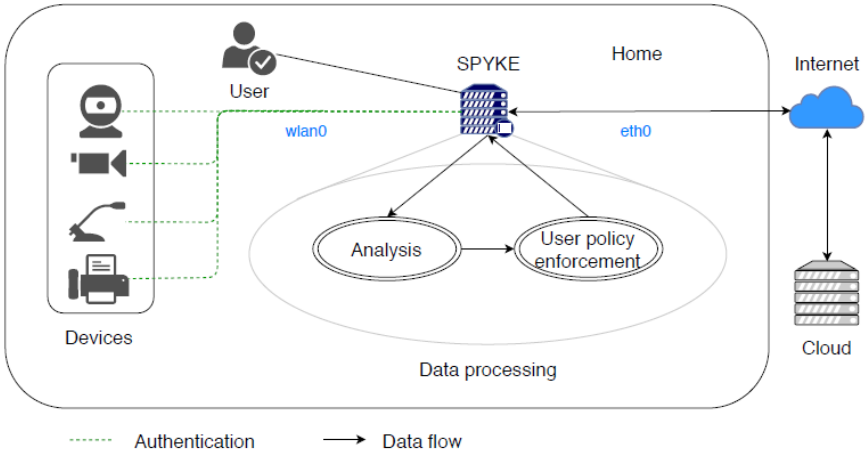

Figure 2.1: SPYKE proposed architecture.

With the objective to accept new devices to the Smart Home network, the authentication is performed on SPYKE via WPA2 protocol and using a password, the user is then able to enforce whitelisting-based *user policy* which provides privacy protection, by selecting which devices have the permission to access the Internet. Every device status is saved on a database, which can set up as "NEW"(by default), "ALLOWED" or "BLOCKED". The traffic in SPYKE is filtered via iptables[1], a very widely used firewall implementation engine in the Linux kernel, with rules defined by the user.

Figure 2.2 represents the data flow of the SPYKE system. First, devices authenticate themselves and obtain an IP address from DHCP server provided by dnsmasq[2]. The engine stores the device information in the database and waits for the user's approval. After the user's approval, the engine adds rules on iptables allowing the access of the device to the Internet, and adds the defined period to the In-Memory data storage that relies on main memory of the computer data storage.

[1]iptables: `https://linux.die.net/man/8/iptables`, accessed on March 12, 2020
[2]dnsmasq: `http://www.thekelleys.org.uk/dnsmasq/doc.html`, accessed on December 19, 2020

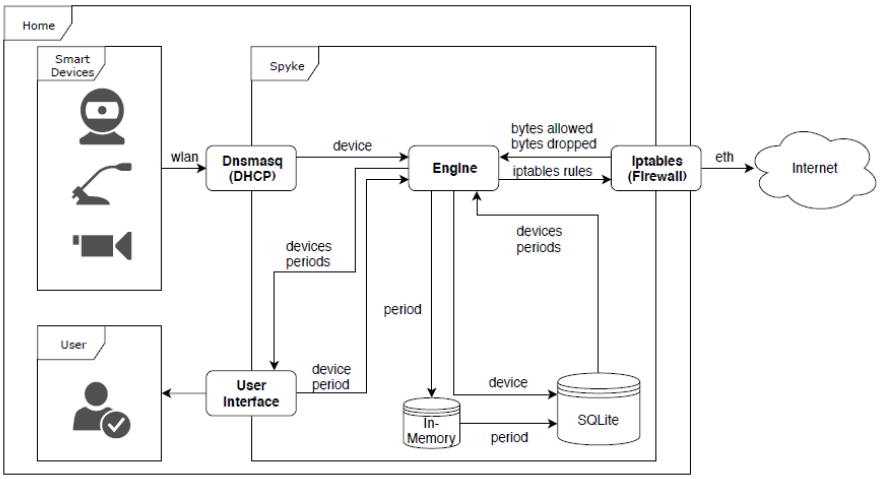

Figure 2.2: SPYKE Data Flow Diagram.

The SPYKE prototype was evaluated in regard to performance and security. This solution was shown to handle a large number of devices rules, which filters outgoing packets with no significant performance degradation.

Furthermore, Bitdefender Box [Ber18] is a commercial security monitor product. Bitdefender Box objective is to secure a Smart Home. As their documentation[3] says, it only requires a small setup to be functional. In addition, on the setup process, it deauthenticates all devices from introduced Wi-Fi and connects to it with the same SSID name and password.

In terms of security, it offers vulnerability assessment to detect network security flaws, exploitation prevention to block attempts to exploit vulnerabilities in connected devices, local device security to protect connected devices in place of a locally installed antivirus, as well as anomaly detection, brute force detection, and data protection. This security monitor also provides an application that can be used outside of the house through the Internet.

2.4 Machine Learning for NIDS

ML can aid security by performing pattern recognition and anomaly detection [CC18]. *Pattern recognition* can discover explicit or hidden characteristics in a given dataset. For example, Deep Neural Networks (Deep NN) have shown promise in outperforming traditional ML techniques for large datasets [DAF18]. However, Deep NN models come with the drawback of being very computationally expensive. Recent works on NIDS using ML include articles by Li et al. and Aung et al.

Li et al. [LZT+19] present two datasets collected from programs involving 126 types of vul-

[3]Bitdefender Box documentation: `https://download.bitdefender.com/resources/media/materials/box/v2/user_guide/BOX_UserGuide_v2_en_.pdf`, accessed on December 19, 2020

nerabilities. The authors conducted a comparative study to quantitatively evaluate the impact of different factors on the effectiveness of vulnerability detection, involving more semantic information, imbalanced data processing, and different neural networks. The experimental results show that control dependency can increase the overall effectiveness of vulnerability detection *f-measure* by 20.3%. However, the imbalanced data processing methods were not effective for the dataset created for the study. Additionally, Bidirectional RNNs (Bidirectional Recurrent Neural Networks) are more effective than Unidirectional RNNs and Convolutional Neural Networks, which in turn are more effective than Multilayer Perceptrons. Finally, using the last output corresponding to the time step for the BLSTM (Bidirectional Long Short-Term Memory) can reduce the False Negative Rate by 2.0% at the price of increasing the False Positive Rate by 0.5%.

Aung et al. [AM18] show the *accuracy* of intrusion detection with the complexity of time when comparing the hybrid algorithm detection method and the single algorithm detection method. The k-Means and Random Tree algorithms were used and each of the methods showed advantages and disadvantages. The experimental results show that the accuracy of the Random Tree algorithm based on k-Means is good in classification of normal and attacks in 10-fold cross-validation but not good in validation. Nevertheless, the model training time of k-Means and Random Trees manifested more suitable time than using single Random Tree algorithm both in 10-fold cross-validation and 66-34 percent validation.

2.5 Machine Learning for IoT and Smart Home IDS

The IoT and the Smart Home are a new challenge for intrusion detection, because instead of just relying on network entities communicating, now there is a diversity of network technologies and protocols involved. Each of the following subsections focuses on related work using a specific Machine Learning approach.

2.5.1 Unsupervised Learning approaches

SPATIO [OM19] was proposed as an anomaly detection system designed for the IoT, based on the stream processing approach. Stream processing is a data processing paradigm in which high-rate data sources are processed and generate results on-the-fly. SPATIO uses a fog computing approach to leverage processing and storage resources of gateway devices, such as routers and base stations in an IoT network to reduce latency and processing cost. SPATIO performs feature extraction, feature selection and outlier detection. For the outlier detection, different type of Unsupervised Learning algorithms were tested, namely: *MCOD*, *ExactSTORM*, *ApproxSTORM*

and *AbstractC*. The authors of SPATIO claim that the system *accuracy* reached close to 80% detection rate in the best scenario. The fog approach showed advantages in both network load and attack detection latency, in comparison with the centralized approach.

Another recent IoT IDS based on Unsupervised Learning models is IoT-NW (Neighbourhood Watch) [CP19]. In this solution, each device sniffs packets in the network and performs feature extraction both at packet and flow level, along with the device states and user presence detection, with the objective to detect malicious interactions between them. The information gathered is used to build behaviour patterns, which allows the detection of abnormal events. The result is a statistical model of the expected values for the RMSE (Root-Mean-Square Error) of each Autoencoder, after running the training phase, for the detection phase to obtain the probability of each network and flow instance being normal, for each device. The authors of IoT-NW claim that their system is capable of detection, but with some delay. The results were sightly improved by inserting contextual information about the user and adjusting thresholds, resulting in a reduction of the False Positive Rate.

2.5.2 Supervised Learning approaches

Kamaraj et al. [KDL19] analysed the potential overhead-savings of ML-based anomaly detection models on the edge, in three different IoT scenarios. The analysis started by first specifying the test-bed, the model creation process and the model benchmarking process. The test-bed included three Raspberry Pi 3 boards, which were used as the edge devices, the cloud, and the interface to connect to an energy measurement platform. Then, it was shown how each anomaly detection model performed on each dataset (scenario) with respect to both overhead and anomaly detection accuracy. The final result showed the best model for each scenario. From the many tested anomaly detection models, the authors asserted that Random Forest, Multilayer Perceptron, and Discriminant Analysis models can viably save time and energy on the edge device during data transmission. Furthermore, it was also asserted that k-Nearest Neighbours (k-NN), although reliable in terms of prediction accuracy, demanded excessive overhead and results in net time and energy loss on the edge device.

Hasan et al. [HIZH19] analysed different types of vulnerabilities detection algorithm in IoT sensors. The performance of several ML models to predict attacks and anomalies on the IoT systems was compared. The ML algorithms that were used in this work were LR (Logistic Regression), SVM (Support Vector Machine), DT (Decision Tree), RF (Random Forest), and ANN (Artificial Neural Network). LR used coordinate descent. SVM and ANN used conventional gradient descent technique. The optimizer was not used in the case of DT and RF because

these are non-parametric models. The final model was evaluated against the testing set using different evaluation metrics. The authors claim that the system obtained 99.4% test accuracy for DT, RF, and ANN. However, although these techniques have the similar accuracy, other metrics showed that RF performs comparatively better. Therefore, the authors concluded that RF was the best technique for this particular study.

Finally, Doshi et al. [DAF18] demonstrated that using IoT-specific network behaviours to inform feature selection can result in high accuracy DDoS detection in IoT network traffic with a variety of ML algorithms. Building on this observation, the authors developed a ML pipeline for data collection, feature extraction, and binary classification for IoT traffic DDoS detection. The features used were extracted from IoT-specific network behaviours, while also leveraging network flow characteristics such as packet length, inter-packet intervals, and type of protocol. Furthermore, a variety of classifiers were compared for attack detection, including RF, k-NN, SVM, DT, and NN (Neural Networks). The classifier training data was generated by simulating a consumer IoT device network, comprised of a router, some consumer IoT devices for benign traffic, and some adversarial devices performing DoS attacks. The authors found that that RF, k-NN, and NN classifiers were particularly effective in identifying attack traffic. The linear SVM classifier performed the worst, suggesting that the data is not linearly separable. The DT classifier performed well, achieving an accuracy of 0.99, suggesting that the data can be segmented in a higher feature space. The k-NN classifier also achieved the same accuracy, suggesting that the two different data classes clustered well in feature space. The NN performed well despite having fewer than half a million training samples from a 10-minute packet capture.

2.5.3 Hybrid Learning approaches

Cramer et al. [CGM+19] described an approach for detecting anomalous behaviour of devices by analysing their event data with few or no numeric characteristics. To perform anomaly detection, the log data passed through feature generation, feature aggregation, and analysis. The first ML approach used was EE (Elliptic Envelope), which is based on fitting an ellipse to the data by assuming that the inlier data are Gaussian distributed. Additionally, the One-Class SVM algorithm was also used, a novelty detection algorithm because it assumes that the training data set is not polluted by outliers or anomalies. Both of these algorithms are, per definition, Supervised Learning algorithms, however, they can be trained as Unsupervised Learning. The EE was trained as Supervised Learning and One-Class SVM as Unsupervised Learning, to compare the results with different approaches. According to the authors, the main benefits of this approach were: the ability of creating an analysis work-flow for specific use

cases, which takes much less time in comparison to using a general-purpose data mining tool; and it is easy to encode domain-knowledge into the analysis work-flow so that the necessary data transformations and feature engineering tasks are made part of the whole analysis.

Xiao et al. [XWL+18] presented IoT security solutions based on a combination of ML techniques including Supervised Learning, Unsupervised Learning, and Reinforcement Learning (RL) for authentication, access control, secure offloading, and malware detection schemes. The authors stated that several challenges have to be addressed to implement learning techniques in practical systems. IoT devices usually have difficulty estimating the network and attack state accurately and have to avoid attacks. A suggested potential solution was Transfer Learning [CC18], which explores existing defence experiences with data mining to reduce random exploration, accelerate the learning speed, and decrease the risks of choosing bad defence policies at the beginning of the learning process. Another challenge was computation and communication overhead. Many existing ML based security schemes have intensive computation and communication costs and require a large number of training data and a complicated feature extraction process. Therefore, new ML techniques with low computation and communication overhead should be investigated. Furthermore, the authors stated that the intrusion detection schemes based on Unsupervised Learning techniques sometimes have non-negligible misdetection rates for IoT systems. Additionally, the Supervised and Unsupervised Learning sometimes failed to detect the attacks due to oversampling, insufficient training data, and bad feature extraction.

Table 2.1: Related work articles comparison.

Article	**Traffic Capture**	**IoT**	**SL**	**UL**	**Machine Learning Algorithms**
Wang et al. [WPC19]	Yes	Yes	No	No	None
Berte et al. [Ber18]	Yes	Yes	??	??	Unknown
Carmo et al. [CP19]	Yes	Yes	No	Yes	ANN - Autoencoders (UL)
Mouta et al. [OM19]	Yes	Yes	No	Yes	MCOD (UL), ApproxSTORM (UL), ExactSTORM (UL), AbstractC (UL)
Doshi et al. [DAF18]	Yes	Yes	Yes	No	Random Forest (SL/ ∼UL), k-NN (SL), NN (SL/ ∼UL), SVM (SL/ ∼UL), Decision Trees (SL/ ∼UL)
Xiao et al. [XWL$^+$18]	∼Yes	Yes	Yes	Yes	Naive Bayes (SL/ ∼UL), k-NN (SL), NN (SL/ ∼UL), Deep NN (SL/ ∼UL), SVM (SL/ ∼UL), Random Forest (SL/ ∼UL), Infinite Gaussian Mixture Model (UL), Q-learning (RL), Dyna-Q (RL), Post-Decision State (RL), Deep Q-network (RL)
Cramer et al. [CGM$^+$19]	No	Yes	Yes	Yes	Elliptic Envelope (SL/UL), One-Class SVM (SL/SS)
Kamaraj et al. [KDL19]	No	Yes	Yes	No	Random Forest (SL/ ∼UL), k-NN (SL), Multilayer Perceptron (SL), Discriminant Analysis Classifier (SL)
Hasan et al. [HIZH19]	No	Yes	Yes	No	Logistic Regression (SL), Decision Tree (SL/ ∼UL), Random Forest (SL/ ∼UL), ANN (SL/ ∼UL), SVM (SL/ ∼UL)
Aung et al. [AM18]	No	No	Yes	No	k-Means (UL), Random Forest(SL/ ∼UL)
Li et al. [LZT$^+$19]	No	No	Yes	Yes	Recurrent Neural Networks (UL/ ∼SL), Convolutional Neural Network (SL/UL), Multilayer Perception (SL) Bidirectional Long Short-term Memory (UL/ ∼SL)

The "??" signifies that the information is unavailable.
The "∼Yes" signifies that the traffic capture is simulated.
Each algorithm is classified by their possible learning types:
(UL) - Unsupervised Learning; (SL) - Supervised Learning;
(SS) - Semi-supervised Learning; (RL) - Reinforcement Learning;
(SL/ ∼UL) - The algorithm is per definition a Supervised Learning algorithm, although technically, they can be trained as Unsupervised Learning for the purpose of clustering;
(UL/ ∼SL) - The algorithm is per definition a Unsupervised Learning method, although technically, they can be trained as Supervised Learning methods, referred to as self-supervised.

2.6 Summary

In this Chapter we started by presenting different Network Defences, where we went more in depth about the Firewall and Intrusion Detection System concept. Then, we introduced Machine Learning and its use for Cybersecurity. Afterwards, we summarized works on anomaly detection mainly applied to IoT environments. Table 2.1 compares the related work according with traffic capture, ML algorithms used and if it is applied to an IoT environment.

The decision to use either a Supervised or Unsupervised ML algorithm depends on factors related to the structure and volume of data and the use case. From the comparison on the Table 2.1, we notice a trend to use Supervised Learning models for anomaly detection in IoT environments. This is an interesting fact, since anomaly detection is strongly related with Unsupervised Learning. Additionally, Unsupervised Learning models are also very adaptive, which means they adjust to and eventually accept changes in the time series when they change to a new normal. The suspected reason for the strong predominance of Supervised Learning is because of the available labeled data from IoT environments, which turn the use of supervised models for anomaly detection more effective. Even though there is a noticeable predominance on Supervised ML, both approaches have potential to be effective for anomaly detection.

Chapter 3

STAKE

In this Chapter we present our proposal, STAKE, which stands for: Secure Tracing of Anomalies using previous Knowledge and Extensions. The *tracing* component refers to the network traffic capture. The *previous knowledge* refers to rules that can be defined to recognise known attacks and enforce limits on device traffic. Finally, the *extensions* refer to the Machine Learning (ML) plug-ins that can be installed and used to detect unknown attacks.

STAKE is a new system, based on the previous SPYKE [WPC19] system, presented in the Section 2.3, where the *extensions* part is novel. We start by presenting with the solution overview and architecture, and then we detail the operation phases: data collection, data pre-processing, anomaly detection and result/output.

3.1 Overview

STAKE is intended to run as a gateway device, located at the Smart Home, between the user IoT devices and the cloud service providers, as represented in Figure 3.1. The Raspberry Pi logo appears on the figure to illustrate the kind of off-the-shelf device where STAKE can run.

Network traffic is constantly changing, especially if new devices are added to the Smart Home or the firmware of existing devices is updated with new features. Thus, a well performing ML model today may no longer be suitable after some time [CC18]. To make the anomaly detection system maintainable, STAKE has a modular system that allows for swapping out, removing, and reimplementing plug-ins. The possibility to update, change or test various type of ML models, which are set up in different kind of plug-ins, is expected to be beneficial in the long run.

For device authentication in a wireless network, we have different security algorithms options to choose from, such as: WEP (Wired Equivalent Privacy), WPA-PSK (Wi-Fi Protected Access with Pre-Shared Key mode) or WPA2-PSK [WPC19]. However, choosing the wrong one, could

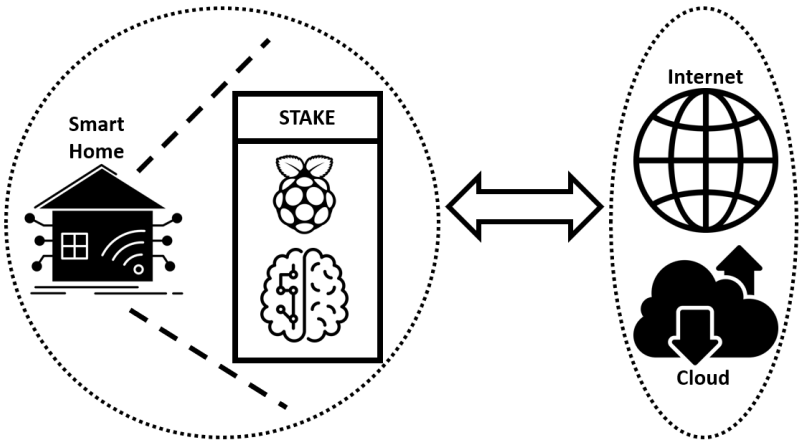

Figure 3.1: STAKE solution overview.

affect the network security. WEP is the oldest and has proven to be vulnerable. WPA improved security in comparison with WEP, but is now also considered vulnerable to intrusion. On the other hand, WPA2 has shown, even though is not perfectly secure, to be the current most secure device authentication option. From the WPA2 security algorithm, there are two different types of encryption to choose from: Temporal Key Integrity Protocol (TKIP) and Advanced Encryption Standard (AES). TKIP is no longer considered secure and is now deprecated. AES, on the other hand, is a more secure encryption protocol. AES itself is a very strong cipher, but the AES-CCMP variation increases the algorithm security by making it more difficult for an eavesdropper to spot patterns, and the CBC-MAC message integrity method ensures that messages have not been tampered with. Taking the previous into consideration, it was decided to use WPA2-PSK algorithm in our system for key management and AES-CCMP algorithm for pairwise cipher.

3.2 Architecture

The STAKE system components are represented in Figure 3.2. The administrator has access to a management graphical user interface via HTTP. The network capturer will intercept traffic of devices, running in batch or stream mode. In batch processing, the system processes all or most of the data at once. In stream processing, on the other hand, the system processes data in a given time window or using only the most recent record. The captured data will be stored in files and in a database. The plug-in manager is able to handle both supervised and unsupervised learning approaches.

In order to avoid loss of packets, processes such as: local HTTP server hosting, data collection (packet capturing), plug-ins training and plug-ins anomaly detection cannot be performed

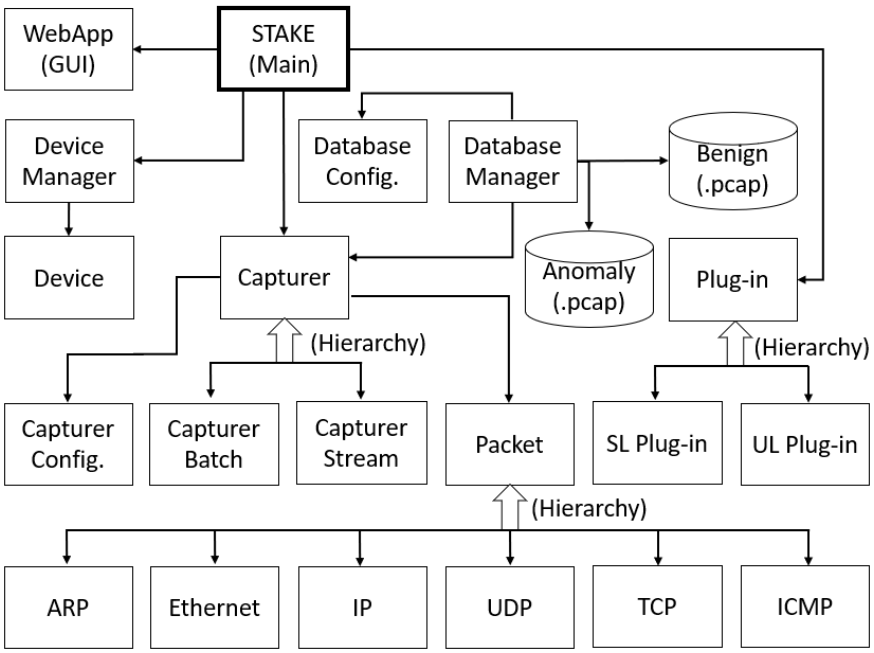

Figure 3.2: STAKE components.

sequentially. For example, if the processes were performed sequentially, the system would need to wait for the plug-ins to finish their training before capturing more packets. Therefore, parallel processing will be required, as represented on the DFD (Data Flow Diagram) in figure 3.3.

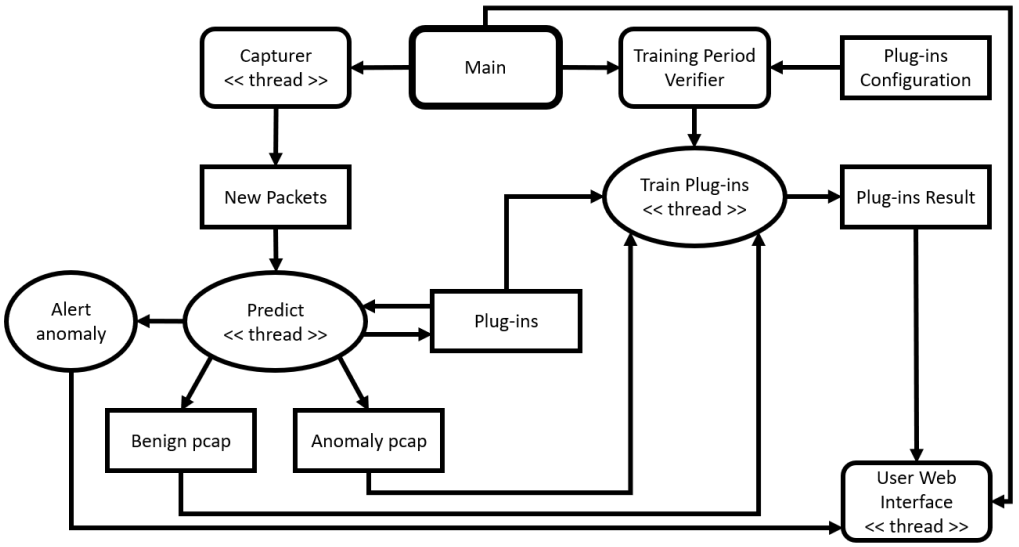

Figure 3.3: Level 0 DFD of STAKE.

The user web interface, except from the template, was built from scratch. After the administrator is authenticated on the system user interface, the administrator is able to: edit his/her account information, identify the network devices according to the associated mac address and add/edit/remove plug-ins. The automatic re-training and anomaly detection function of the

plug-ins are disabled by default. When adding or editing plug-ins, the administrator is able to: enable/disable plug-in anomaly detection, enable/disable automatic re-training, select the re-training periodicity, choose the anomaly percentage of the data, select the amount or time period of the data, select data transformation options, choose the features and choose the evaluation methods. Additionally, the administrator has access to information such as: captured packets statistics, system hardware status information (hard disk memory usage, CPU usage, RAM memory usage and network bandwidth usage), system log, anomalies alerts, plug-in status (enabled, disabled, training or error) and plug-in training results. The plug-in training results helps the administrator to confirm if the plug-in is well configured or requires further adjustment.

Figure 3.4 presents a screenshot of the STAKE solution with the administrator dashboard. The dashboard displays a summary of information available from the other pages, such as: Captured Traffic page, System Information page, Device Management page, Alerts page and Plug-ins Results page.

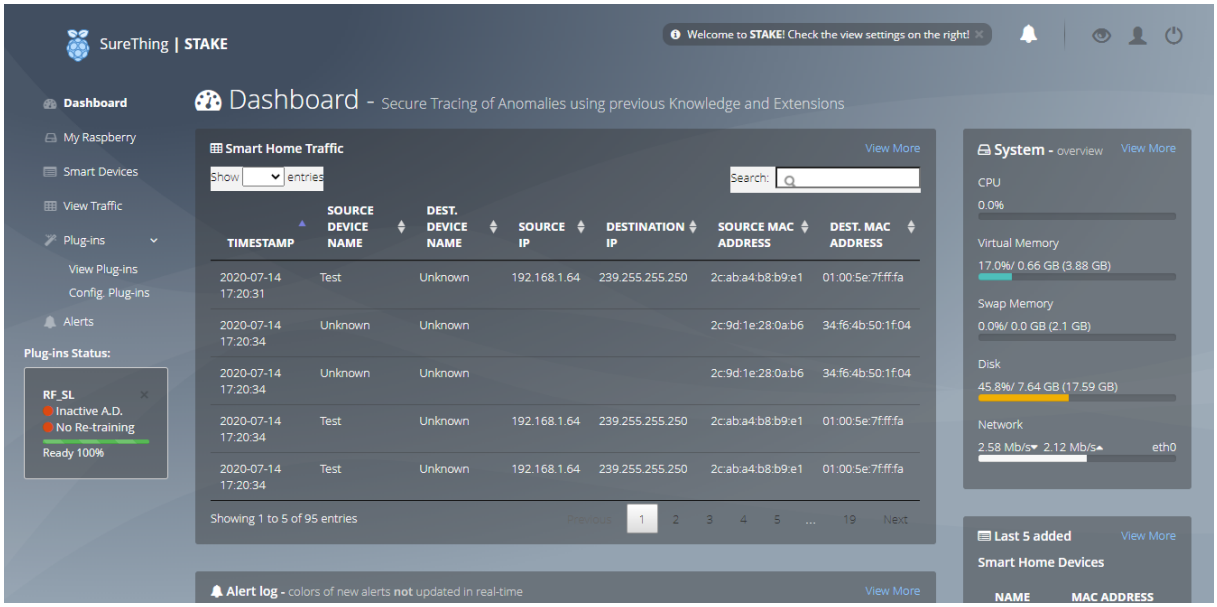

Figure 3.4: STAKE dashboard of the Web interface.

When configuring the plug-ins, the administrator has the option to select the samples used for training according to the quantity of samples or according to the time period of the samples. By selecting the quantity of samples option, the last n samples from both benign and anomaly datasets are read for training. On the other hand, by selecting the time period of samples option, the samples selection is performed according the chosen time period by the administrator. This period selection can be done from seconds to months. Both samples selection options should be performed attentively when the system is recently implemented, since the selected number of samples or time period of samples could not match with the existing samples in the datasets.

In addition to the system settings and information that the administrator has access after the authentication (stored in the PostgreSQL[1] database), the administrator also has a system configuration file, saved as ".ini" extension, for configurations that are necessary to configure before implementing and starting the system. These initial system configurations located in the configuration file are settings related to the web server cookies, connection to the database located in postgreSQL, file paths (dataset, logs and plots), network adapter settings, IP configuration and system port configuration that the server will be hosting.

STAKE operates in four main phases: data collection, data pre-processing, anomaly detection and result/output. The last three phases are executed by plug-ins using models. Each ML model is trained with data from the captured traffic archives, with the objective to detect device operation anomalies, and to do so in near real-time. By "near real-time" we mean that the data processing may be slightly slower than real-time, but the anomaly detection alarms should still be raised in a timely way for security purposes [DD11].

3.3 Data Collection

In the first phase, STAKE captures incoming and outgoing traffic in different detail levels: network traces, network flows, and summary features. This data is stored in a persistent repository with adequate indexing for retrieval e.g. by device, by time range.

Network trace contains whole communication between IP devices over LAN, a network flow is a series of communications between two endpoints that are bounded by the opening and closing of the session, and summary features is the representation of network traces according to the specific selected information.

Examples of well known capture software are tcpdump[2] and Wireshark[3]. The capture operation of the STAKE infrastructure is handled by a libpcap[4] tool, known as Scapy[5]. During collection, real-time parsing of the capture file is needed. The parser extracts statistics, information from the packets and store the summary of the packets in a database. Packets received by the main server are then appended to a file with the pcap extension.

The pcap file reading was first tested with Scapy as well. However, a faster alternative was found. The dpkt[6] tool demonstrated to be more efficient and notably faster when reading packets in comparison to Scapy. Nevertheless, in terms of packets capturing, Scapy performed

[1]PostgreSQL: `https://www.postgresql.org/`, accessed on September 29, 2020
[2]tcpdump: `http://www.tcpdump.org/`, accessed on March 15, 2020
[3]Wireshark: `https://www.wireshark.org/`, accessed on March 15, 2020
[4]libpcap: `http://www.tcpdump.org/pcap.html`, accessed on March 15, 2020
[5]Scapy: `https://scapy.readthedocs.io/en/latest/introduction.html`, accessed on March 15, 2020
[6]dpkt: `https://dpkt.readthedocs.io/en/latest/index.html`, accessed on June 20, 2020

faster.

Both sampling and filtering was considered when handling data. Unfortunately, inadequate sampling can miss packets that would identify or describe an incident, and filtering without a good understanding of the packets can be detrimental in a way that analysts will not be able to understand incidents [DCL18]. Filtering can be implemented in-line or post-processing. Post-processing takes place after the storing of the captured traffic and in-line takes place during the actual traffic capture process [DCL18]. For STAKE, we opted for post-processing filtering, performed individually by each plug-in, since each plug-in has its own data pre-processing requirements.

An additional consideration for data collection was to decide if the system will operate in batch processing or stream processing. In a situation where timely decision-making is required, a stream processing approach should be considered. However, in situations where time is not a critical factor in decision-making, batch ML algorithms can be performed. Since STAKE aims for a near real-time detection of device operation anomalies, we opted mainly with a stream processing approach. Nevertheless, batch processing will also be available for plug-ins whose ML models require this kind of processing.

The last consideration was the benign and anomaly packets segregation, as shown in the figure 3.3. Labeled data is essential for the plug-ins to be evaluated and for the supervised plug-ins to be trained. Two solutions were taken into account: store all the packets on a single pcap file and have the packet labels in a separated file or segregate both benign and anomaly packets into two different pcap files. The second solution was considered the most efficient and simpler one. Additionally, by segregating both types into different files, it was considered that false negatives and positives analysis would be an easier task.

3.4 Plug-in types and processing phases

STAKE allows one or more plug-ins to be installed. Each plug-in can use the collected data to train a ML model, and then use it to detect anomalies. Figure 3.5 provides an overview of the operation phases, sub-phases and steps for the plug-ins. On the left-side, we see the phases for the Supervised Learning, and on the right-side, for the Unsupervised Learning. The data cleaning and data selection steps are not required, so they are greyed out. The following sections will detail these phases: Data Pre-processing, Anomaly Detection and Result/Output.

Supervised Learning Plug-in	Unsupervised Learning Plug-in
1. Data Preprocessing 1.1. Feature Extraction 1.1.1.Data Integration 1.1.2. Data Cleaning 1.2. Feature Selection 1.2.1. Data Selection 1.2.2. Data Transformation 2. Anomaly Detection 2.1. Data Mining 2.2. Pattern Evaluation 3. Result/Output	1. Data Preprocessing 1.1. Feature Extraction 1.1.1.Data Integration 1.1.2. Data Cleaning 1.2. Feature Selection 1.2.1. Data Selection 1.2.2. Data Transformation 2. Anomaly Detection 2.1. Data Mining 2.2. Pattern Evaluation 3. Result/Output

Figure 3.5: Plug-in types comparison.

3.5 Data Pre-processing

An important phase for Supervised Learning algorithms is data pre-processing [DD11], because it allows for improvement in the quality of the data that is used. Data pre-processing has a significant impact on the performance of Supervised Learning models, since unreliable samples probably lead to wrong outputs. On the other hand, some of the Unsupervised Learning models have the the ability to pre-process the data in the anomaly detection phase. Thus, this phase is not always needed for this type of ML models [CC18].

Anomalies may be undetectable at one level of granularity, or abstraction, but easy to detect at another level [DD11]. One of the main challenges is to choose the features that best represent the user or the system behaviour patterns so that anomalous behaviour will be detected, whereas benevolent behaviour will not be wrongly classified as anomalous. Therefore, considering the complexity of this phase, it was decided to divide the phase into Feature Extraction and Feature Selection.

3.5.1 Feature Extraction

Feature Extraction can be done in two steps: data integration and data cleaning. The data integration step consists in combining data from multiple and heterogeneous sources into one database. This first step is required on every ML model category, to simplify the data training process. Considering the fact that the objective of Unsupervised Learning is to detect outliers, the second step should only be done when using Supervised Learning algorithms. Removing the noise would consequently remove the Unsupervised Learning algorithms purpose of anomaly detection.

A feature extractor derives basic features that are useful in event analysis engines. The

features chosen for extraction to train the model depend on the context of the problem. In building an anomaly detection system, it is important to consider whether the data used to train the initial model is contaminated with anomalies. Data cleaning, also known as data cleansing, removes noise and irrelevant data from the collected data [HIZH19]. However, cleaning a dataset with the intention to remove anomalies is risky, since anomalies can turn out to be a new pattern behaviour of a benign device. In these cases, the best option is to start off by accepting that the data contains anomalies and iteratively move towards a better solution.

Using unbalanced datasets can result in a bias that is difficult to detect [CC18], such as selection bias and exclusion bias. Selection bias is when proper randomization is not achieved, thereby ensuring that the sample obtained is not representative of the device behaviour intended to be analysed. On the other hand, exclusion bias may also exist, which results from exclusion of benevolent device behaviour from the sample, e.g. exclusion of new benevolent device behaviours which have recently been performed in the Smart Home network.

Likewise, training a model with a dataset that has a lot of missing values can also drastically impact the ML model quality. The main solutions for missing values are removing any event with missing features or impute the value of the missing feature. This impute of missing values can be done with the mean, median, or most frequently appearing value (mode) of the column.

3.5.2 Feature Selection

Feature Selection should only be performed in Unsupervised Learning models when data normalization or data dimensionality reduction is beneficial. It consists in reducing the number of random variables under consideration by obtaining a set of main variables.

This sub-phase is done in two steps: data selection and data transformation. The selection of features is expected to reduce overfitting probability, improve model prediction *accuracy* and reduce training time. Overfitting occurs when the model or the algorithm fits the data too well. On the other hand, underfitting occurs when the model or the algorithm does not fit the data well enough [CC18]. Data selection allows the user to obtain a reduced representation of the data set to keep the integrity of the original data set in a reduced volume. Having a number of features greater than the number of data points will make the ML model overfit. Thus, for better model performance, down-sample of large-scale events may be needed.

Regardless of the model being used, it is recommended to use regularization parameters based on experimental data. Regularization is a technique which makes slight modifications to the learning algorithm such that the model generalizes better. This in turn improves the performance of the model on the unseen data as well.

Data transformation is when the selected data is transformed into suitable formats [CGM+19]. Before data can be processed within ML models, there are certain data transformation steps that must be performed. These transformations include changing data types, removing string formatting and non-alphanumeric characters, converting categorical data to numerical and converting timestamps. Additionally, normalization is a technique often applied as part of data transformation for ML models. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. Even though not all datasets require using this technique, it should be used when features have different value ranges.

3.6 Anomaly Detection

This phase enables the ability to classify and detect intrusions on a network, such as a Smart Home. An anomaly detection system is considered optimal when [DD11]: returns a low count of False Positives and of False Negatives; is easy to configure, tune and maintain; adapts to changing trends in the data; resource-efficient and suitable for real-time application.

In case of using a Supervised Learning model plug-in, it learns from prior data on the first step and makes a prediction about the future behaviour on the second step. Taking this into consideration, when using Supervised Learning this phase is separated into two steps: data mining and pattern evaluation. In the data mining step, the analysis tools are applied to discover potentially useful patterns. The upcoming step, pattern evaluation, useful patterns are identified using given validation measures. On the other hand, an Unsupervised Learning model plug-in is able to perform the anomaly detection in a single step.

Before initiating the training phase, hyper-parameters are typically chosen. Hyper-parameters are a fragile component of ML systems because their optimality can be affected by small changes in the input data or other parts of the system [CC18]. Therefore, an efficient hyper-parameter optimization should be considered. The optimal hyper-parameter configuration can be found by providing a fitting metric for comparing each classifier performance with different possible combinations of hyper-parameter values. Even for algorithms that have only a few hyper-parameters, grid search, an exhaustive sweep through the hyper-parameter space of a ML algorithm, is a very time and resource-intensive way to solve the problem because of combinatorial explosion [CC18].

3.7 Result/Output

When using various types of ML plug-ins, having a unified location for reporting and alerting can make a difference in the value of security alerts raised. The auditing of alerts raised by an anomaly detection system is important, since it enables the ability to evaluate the system, as well as investigating False Positives and Negatives [CC18]. Therefore, alerts will be stored in a log file and the packets information will be stored in the benign or anomalous pcap files.

Developing a system which returns explainable results is critical to build trust in the ML system. The system is explainable if it presents enough information to allow the user to derive an explanation for the decision. Thus, the alerts provides information about the source and destination devices of the detected anomalous packet and which plug-in detected the anomaly.

The possibility of the ML security systems being themselves attacked must also be taken into account. Common attacks on this kind of system are model poisoning and model evasion. Systems that continually learn from input data and instantaneously feedback labels which are provided by users, are vulnerable for model poisoning, in which the attacker inject intentionally misleading data to skew the decision boundaries of the model classifiers. Furthermore, model evasion implies that the attacker evade classifiers with adversarial examples that are specially crafted to trick specific model and implementations. Using robust statistical algorithms which are resilient to poisoning and probing attempts is a way of slow down the attacker. This can be done by maintaining test sets and heuristics that periodically test for abnormalities in the input data, model decision boundary, or classification results [CC18].

3.8 Summary

In this chapter, we presented our solution, STAKE, that will be able to: provide an execution environment with plug-ins for anomaly detection, using ML models trained from the captured data archives, and execute near real-time detection of device operation anomalies. The phases, sub-phases and steps will be different depending on the ML model category.

Chapter 4

Evaluation Methodology

In this Chapter we present our evaluation methodology. We start by discussing the system performance evaluation, the hardware to be used, and the dataset evaluation metrics, considerations and description. Additionally, we also discuss the exemplary plug-ins that will demonstrate the functionalities of STAKE, and the plug-ins specific evaluation metrics.

Overall, the methodology is the following: create a test-bed with diverse Smart Home equipments, deploy the STAKE software in the gateway, collect significant and representative datasets, assess performance, develop exemplary plug-ins and confirm that these plug-ins are effective at anomaly detection, showing that the STAKE platform supports the execution of diverse plug-ins. The plug-ins results are illustrative of what can be achieved with the STAKE platform, and they show by example, that more and better plug-ins can be developed by the community.

4.1 STAKE Performance Evaluation Metrics

The metrics chosen to test the system were focused on computational and time performance. We were interested in answering the following questions about our system:

- "How much of the device CPU (%) is used when no plug-ins are training, when training each plug-in individually and when training both plug-ins simultaneously?";
- "How much of the device RAM memory (%) is used when no plug-ins are training, when training each plug-in individually and when training both plug-ins simultaneously?";
- "What is the average delay between the moment that the packet is captured and when it is reported as an anomaly via an alert from each plug-in when no plug-ins are training, when training each plug-in individually and when training both plug-ins simultaneously?";
- "What is the difference between the time duration of the model training when creating

each plug-in and the duration of the model training of each plug-in when implemented on the system?";

- "Does the plug-ins have the same scoring results when implemented and re-trained in STAKE system?";

From these metrics, we aim to compare all the possible situations and observe how to system reacts in each one of them.

4.2 Hardware

We implemented the system on a Raspberry Pi 4 model B (figure 4.1), with Raspbian operating system, installed in a 128 GB SD card. This small single-board computer provides a network interface card with 2.4 GHz and 5.0 GHz IEEE 802.11ac WLAN (*wlan0* adapter) protocol and a network interface via cable with Gigabit Ethernet (*eth0* adapter) up to 300 Mbps.

Figure 4.1: Raspberry Pi 4 model B.

Using a Raspberry Pi as an IoT gateway has the benefit of having the ability to build programs that can use other services and third-party libraries. Additionally, not only many access points have lower CPU power and RAM size compared to Raspberry Pi, but also access points with similar CPU values are more expensive than the Raspberry Pi. Furthermore, Raspberry Pi 4 model B has the benefit of possessing a QuadCore processor (Broadcom BCM2711, Quad core Cortex-A72 64-bit SoC @ 1.5GHz), which allows us to process threading efficiently.

For storing the collected packets, it was used the available free memory in the mentioned 128 GB SD card. However, other possible considered alternatives were Hard Disk Drives (HDDs) and Solid State Drives (SSDs).

The Smart Home network used for dataset creation consists in devices beyond the usual standard devices. The devices that constitute the network are: a Smart Plug model TP-Link-HS110, a Raspberry pi 4 Model B 4GB with camera model v1 for video streaming, a Raspberry pi 4 Model B 4GB to implement STAKE system, two Android smart phones, one Android

tablet, a laptop with Windows 10 Pro operating system and a malicious laptop with Kali 2019.3 operative system to perform attacks in the network.

4.3 Dataset

The dataset is one of the critical elements to be used on anomaly detection systems. Datasets contain information about how systems can learn normal behaviour patterns, in order to identify abnormal behaviour. Since we are interested in evaluating STAKE behaviour in a Smart Home environment, we decided to capture our own set of data from a real smart home network.

4.3.1 Evaluation Metrics

An efficient measure for dataset evaluation is analysing the dataset feature correlation, which allows us to discover how strong a relationship is between features. The selected correlation coefficient was Pearson method (equation 4.1).

$$\text{pearson} = \frac{cov(X,Y)}{\sqrt{var(X) * var(Y)}} \tag{4.1}$$

Given a pair of random variables (X, Y), where cov = covariance and var = variance

The correlation coefficient ranges from -1 to 1. A value of 1 implies that a linear equation describes the relationship between X and Y perfectly. A value of -1 implies that all data points lie on a line for which Y decreases as X increases. A value of 0 implies that there is no linear correlation between the variables.

4.3.2 Considerations

During the dataset evaluation, we took into consideration that the data may start out with one particular pattern and then, after a period of time, change into a totally different one [DCL18].

An additional important considered aspect was whether the data has *seasonal patterns*. A seasonal pattern exists when a series is influenced by seasonal factors (e.g. the quarter of the year, the month, the hour of the day or the day of the week) [DCL18]. Assuming seasonal patterns have two major drawbacks: it may require too many data points to obtain a reasonable baseline and it may produce a poor normal model. It was also considered that if we assume there are no seasonal patterns in any of the metrics and we apply standard techniques, we are either going to be very insensitive or too sensitive to outliers, depending on which technique we are using. This highlights the need for adaptive learning algorithms with the ability to learn the model with every new data captured.

Updating the model with every captured packet (including abnormal ones), is one possible strategy, but it is not considered a good one. If an outlier persists beyond that captured packet, the model will start considering that outlier as a normal behaviour. Therefore, there should be a balance in how fast we learn versus how adaptive we are towards the dataset.

Each captured packet in the dataset should address topics, through at various levels of detail and represented as features, activities concerned with who or what is communicating over the network, how those communications are taking place and the purpose of those communications.

Last consideration was feature correlation. Correlation is a way to understand the relationship between multiple features in the data. Features with correlation coefficient over 0.95 are usually considered as highly correlated features. If the dataset has highly correlated features, then there is a high chance that the performance of the model will be impacted by a problem called "multicollinearity". Multicollinearity happens when one feature in a model with linear classification approach can be linearly predicted from the other features with a high degree of accuracy. This normally leads to skewed or misleading results. However, non-linear models are not affected by this problem.

4.3.3 Description

The network was captured during a period of one week, between 30/06/2020 19:23:04 and 07/07/2020 19:22:04. This captured dataset consists in a total of 1158539 benign samples (labeled as '1') and a total of 93778 anomaly samples (labeled as '-1'). In other terms, from 1252317 captured samples, 92% are benign and 8% are anomalies.

This set of anomaly samples contains a variety of network attacks performed by the malicious laptop, such as: port scanning, TCP SYN flooding, ICMP flooding and ARP spoofing. The devices that had their ports scanned were: STAKE, the Raspberry Pi video streamer and the Windows 10 Pro laptop. The SYN and ICMP flooding victim devices were: STAKE and Raspberry Pi video streamer. At last, the ARP spoofing attack was performed with the Raspberry Pi video streamer as the victim device.

Regarding the protocols, every protocol is captured and stored in the database. However, STAKE system only considers the following packets: Ethernet, IP, TCP, UDP, ICMP and ARP. The protocols and features were selected according to the information that they offer for anomaly detection In other words, the features in which attacks cannot be performed are excluded for analysis.

The first selected protocol was the Ethernet, because this protocol is commonly used in local area networks. This protocol describes how the networked devices share their data through the

physical medium. The IP is an important protocol for network analysis, because it provides end-to-end datagram transmission across multiple IP networks. The TCP was selected because it is the dominant protocol for a majority of internet connectivity. Additionally, the UDP was selected because it is a communications protocol that enables process-to-process communication and runs on top of IP. The UDP protocol improves data transfer rates over TCP and best suits applications that require lossless data transmissions. The ICMP protocol is commonly used for error reporting and to perform network diagnostics, therefore it considered for anomaly detection. At last, the final selected protocol was the ARP, which maps an IP address to a MAC address that is recognized in the local network.

The feature selection on all the protocol had the same approach. The main approach was to only select the packet fields that could give information of an anomalous devices behaviour in an IoT environment, while excluding the packet fields that return same, similar or no useful information for anomaly detection.

The selected features from the Ethernet protocol were: source Ethernet port, destination Ethernet port and Ethernet type field. No features were excluded on this protocol.

The selected features from the IP protocol were: source IP address, destination IP address, encapsulated protocol type, header length, 'do not fragment' flag, 'more fragments' flag, 'reserved' flag, , fragment offset value, time to live (TTL) and type of service (ToS). The excluded features from the IP protocol were: version, options field, 'fragment offset' flag, checksum and identification field.

The selected features from the TCP protocol were: sequence number, acknowledgement number, source port, destination port, packet flags, data offset value, urgent pointer flag and window size. The excluded features from the TCP protocol were: options field and checksum.

The selected features from the UDP protocol were: source port, destination port. The excluded features from the UDP protocol were: checksum.

The selected features from the ICMP protocol were: type and code (subtype) of the icmp message. The excluded features from the ICMP protocol were: checksum.

The selected features from the ARP protocol were: hardware type, sender hardware address, target hardware address, target protocol address, target protocol address and operation code. The excluded features from the ARP protocol were: hardware address length and protocol type.

At last, the extra selected features were: packet payload, packet length and timestamp.

The features from the captured data that showed high correlation (pearson correlation coefficient over 0.95) were the ARP features with the IP type field feature and between the TCP fields. This was expected, since the mentioned features return the same information. However,

none of them were excluded because attacks can be performed on a single feature.

4.4 Exemplary Plug-ins

To demonstrate that STAKE is a functional executing environment for anomaly detection plug-ins, we developed two exemplary plug-ins. We decided to implement an *Elliptic Envelope* based plug-in and a *Random Forest* based plug-in. This plug-in choice was influenced by algorithms chosen by other works in the literature [HIZH19, DAF18, XWL+18, CGM+19]. From the related work Table 2.1, it is noticeable the trend for *Random Forest* algorithm approach in Smart Home environment. On the other hand, even though *Elliptic Envelope* is not that frequent as *Random Forest*, it has shown interesting results, which spiked curiosity for further investigation.

4.4.1 Plug-in Evaluation Techniques

Regarding evaluation techniques, we took into consideration the use of cross-validation to evaluate the plug-in. Cross-validation is a standard method model evaluation in situations where there is not a lot of training data, so every labeled example is able to make a significant contribution to the learned model [CC18]. This type of evaluation method is commonly used when the goal of the algorithm is prediction. In this method, the labeled data is divided into k equal parts and it is trained k different models. Afterwards, each model "holds out" a different one of the k parts and trains on the remaining k–1 parts. The held-out part is then used for validation. Finally, the k different models are combined by averaging both the performance statistics and the model parameters [LZT+19].

Additionally, grid search was also a considered evaluation technique. Grid search is a tuning technique that attempts to compute the optimum values of hyper-parameters. It is an exhaustive search that is performed on the specific parameter values of a model. The model is also known as an estimator.

4.4.2 Elliptic Envelope

The Elliptic Envelope (EE) is an Unsupervised and Supervised algorithm, where data is distributed across an ellipse, hence the name [EA19]. In our solution, we decided to develop this model with an Unsupervised Learning approach, because we are interested to observe if the anomalies can be detected in an unlabeled training data setting. This algorithm routine models the data as a high dimensional Gaussian distribution with possible co-variances between feature dimensions. In other words, it attempts to find a boundary ellipse that contains most of the normal distributed data. Any data outside of the ellipse is considered to be an anomaly. For

distributed datasets, EE fitting can be a simple and elegant way to perform anomaly detection. More likely than not, this algorithm should be suitable in anomaly detection problems with the time dimension excluded [CC18]. Applying an EE-based plug-in in a streaming anomaly detection system is straightforward. This can be achieved by periodically fitting the EE to new data. The system will have to constantly update decision boundary with which to classify new data points.

The EE model is categorized as a multivariate anomaly detection technique. Multivariate anomaly detection techniques take input from all the devices together as one, without separating them. The most noticeable downsides in using multivariate anomaly detection techniques is the scalability and it is often hard to interpret the cause of the anomaly. These type of techniques are best used with just several hundred or fewer metrics [DCL18].

The EE algorithm has the advantage of using a robust covariance estimator such as the MCD (Minimum Covariance Determinant), which minimizes the impact of training data outliers on the fitted model. MCD is able to discriminate between outliers and inliers, generating a better fit that results in inliers having small distances and outliers having large distances to the central mode of the fitted model. EE is known to fit reasonably well in a two-dimensional contaminated dataset with a known Gaussian distribution, but not so well on a non-Gaussian dataset [CC18]. Additionally, EE are also known to work better on datasets with low dimensionality. When fitting a high dimensional multivariate Gaussian distribution, the data is structured as an hyper-ellipsoid. This increases the complexity of anomaly detection, since anomalies are detected by a combination of features. In some scenarios, the recommended practice is to remove time from the feature set and just fit the model to a subset of other features. However, in these cases, there are risks that the outliers might not be detected.

With this EE Unsupervised Learning based plug-in, we aim to demonstrate that STAKE supports the necessary calls for an Unsupervised Learning algorithm that uses multivariate anomaly detection technique, excludes time dimension and requires training data to be Gaussian distributed. Obtaining good results with this model will demonstrate that the anomalous and benign traffic is distinct enough for it to be fitted efficiently in the model. On the other hand, if the model is not able to fit the data efficiently, it will prove that the captured data is not distinct enough, in other words, benign and anomalous data are excessively mixed inside and outside the generated ellipse or hyper-ellipsoid (depending on the dimension of the data).

4.4.3 Random Forest

The second exemplary plug-in was selected to be distinct for the first one. The Random Forest (RF) classifier is, per definition, a Supervised Learning algorithm, although technically, it can be trained as Unsupervised Learning with the purpose of clustering. In case of our solution, we decided to develop this model with an Supervised Learning approach. Even though Unsupervised learning methods are mostly preferred over Supervised Learning methods in most cases for anomaly detection [CC18]. We considered that researching a Supervised Learning algorithm behaviour in the Smart Home environment would be interesting to observe.

Tree-based models, such as DT and RF, tend to have very good prediction performance. The reason for this good prediction performance is due to the fact that every query interacts only with a small portion of the model space. Nevertheless, single DT tend to overfit to their training sets and, on the contrary, RF mitigates this effect by taking the average of multiple DT, which consequently improves the model performance.

Even though the RF classifier is known to fit real-world data effectively [CC18], the algorithm is black-box in relation about the decision making processes, meaning that these processes are completely opaque to an external observer. A noticeable strength from the RF classifier, is that it can be parallelised to a high degree [CC18].

Given that each randomized DT (Decision Tree) that makes up the forest is independently created and can be individually queried for the generation of the final prediction, this conversely makes the classifier strongly scalable.

The RF classifier have shown better prediction *accuracy* on large data sets with high dimensional feature space [CC18]. This algorithm trains a model by iterating through data points in the given training set, randomly selects a feature and randomly selects a split value between the maximum and minimum values of that given feature across the entire dataset [CC18]. The intuition behind this method is that inliers have more feature value similarities, which requires them to go through more splits to be isolated. Outliers, on the other hand, are theoretically easier to isolate with a small number of splits, since they will presumable have some feature value differences that distinguish them from inliers. In other words, anomalous data points have tendency to have shorter path lengths than regular data points.

In comparison with the EE algorithm, the RF classifier can be applied on a non-Gaussian anomaly contaminated dataset. However, it must be taken into account that we must avoid using very low-dimensional data with the RF classifier for anomaly detection, since it might not be suitable because of the small number of features on which we can perform splits, which consequently can limit the effectiveness of the algorithm. Furthermore, in contrast to EE,

RF can be categorized as an univariate or multivariate anomaly detection technique. In the univariate anomaly detection category, the system looks at each metric by itself, learning its normal patterns and yielding a list of anomalies for each single metric. Often, in univariate algorithms, it is difficult to perform root cause analysis of an issue because it is hard to see the forest for the trees. The advantage of this category is that it is easier to build than other type of methods. Not only the scalability, in terms of computation, is better, but also less data is needed to learn normal device behaviour since the system looks at each metric by itself, as opposed to looking at combinations of metrics. However, one of the drawbacks is that, when something unexpected happens, it affects a lot of metrics, an the system yields a "storm" of anomalies.

4.4.4 Specific Plug-in Evaluation Metrics

Apart from the general Machine Learning evaluation metrics from the Section 2.2.2, specific evaluation metrics for the selected plug-ins were considered. For EE, it was performed a quality measurement of the plug-in fitted model by calculating the distance between outliers and the model's distribution, using a distance function such as *Mahalanobis* distance. It is an useful metric that measures the distance between a point (vector) and a distribution. This metric has applications in multivariate anomaly detection, classification on highly imbalanced datasets and one-class classification. This distance function transforms the columns into uncorrelated variables, scale the columns to make their variance equal to 1 and, finally, calculates the *Euclidean* distance (equation 4.2).

$$\text{Euclidian Distance} = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{4.2}$$

In case of the RF plug-in, the MSE (Mean Squared Error) (equation 4.3) measurement is commonly used, or just a SSE (Sum of Squared Errors) (equation 4.4), which represents the differences between the observed and predicted value.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y}_i)^2 \tag{4.3}$$

$$\text{SSE} = \sum_{i=1}^{n}(y_i - \overline{y}_i)^2 \tag{4.4}$$

4.5 Summary

In this Chapter we presented the evaluation methodology. We started by discussing the evaluation methodology regarding the system performance. In addition, we discussed the Prototype Hardware to be used in the evaluation of this solution. Afterwards, we discussed the dataset evaluation metrics, considerations and description and how it could affect the final evaluation of the STAKE system. At last, we discussed the two selected Exemplary Plug-ins, and the plug-ins specific evaluation metrics.

The selected Elliptic Envelope and Random Forest plug-ins differ in many different aspects, such as: model category (Supervised vs Unsupervised Learning), type of expected input data (Gaussian vs non-Gaussian), efficiency in different dimensional data (Low dimensional vs High dimensional). In addition, while discussing the plug-in evaluation, we concluded that having a variety of different types of plug-ins, has the benefit of providing different metrics of evaluating an anomaly. Therefore, this will allow us to analyse the best Machine Learning approach given the specific environment.

Chapter 5

Results

In this Chapter, we start by discussing the experiments involved the STAKE prototype performance evaluation in terms of computational and time resource. Afterwards, we present the Elliptic Envelope and Random Forest plug-ins creation, and the evaluation methodology criteria used to optimize the models used in the plug-ins. During each plug-in creation and evaluation, it was taken into account: the plug-ins different types of behaviour, specific machine learning algorithm behaviour, general and specific machine learning algorithm evaluation methodologies and all the dataset engineering considerations. At last, we discuss the training results comparison when creating each plug-in and the training results when the plug-ins were implemented and re-trained in STAKE.

In Section 5.1 we analyse STAKE prototype performance. In Section 5.2 we present the results of the Elliptic Envelope plug-in creation and evaluation. Subsequently, in Section 5.3 we present the results of the Random Forest plug-in creation and evaluation. At last, in Section 5.4 we compare the training results of the plug-ins before and after implementing in the system.

5.1 STAKE

This set of testing involved the designed STAKE system architecture and its developed prototype. First we evaluated the system and hardware performance, and then we compared the plug-ins training duration from plug-in creation with the plug-in training duration when implemented in the system.

5.1.1 Performance Evaluation

For the STAKE system evaluations, it was considered that registering only one value would not be a correct evaluation of the system performance. Thus, the evaluation was performed by an

average of 10 values during different periods of time while training different combinations of plug-ins. These combinations were: no plug-ins being trained, each plug-in trained individually (Elliptic Envelope and Random Forest respectively) and both plug-ins trained simultaneously.

From the results from Table 5.1, we can firstly observe that the processor performance is not significantly affected when training both plug-ins simultaneously. Thus, it is possible to conclude that the processor is not heavily affected in threading two plug-ins simultaneously. On the other hand, RAM memory usage is noticeable increased. This observation was expected, since each plug-in training is threaded, each plug-in needs to store the training data individually.

Table 5.1: STAKE average of 10 tests results performance evaluation. EE = Elliptic Envelope and RF = Random Forest.

Component Evaluated	**Plug-ins being Trained**	**Average Value**	**Minimum Value**	**Maximum Value**
Processor Usage	None	18,45%	0,0%	28,10%
	EE	38,80%	25,20%	51,40%
	RF	33,10%	25,00%	50,70%
	EE & RF	39,85%	27,50%	51,00%
RAM Memory Usage	None	19,45%	13,70%	23,70%
	EE	31,15%	22,40%	31,70%
	RF	37,35%	31,20%	38,40%
	EE & RF	48,35%	43,50%	53,00%
Elliptic Envelope Anomaly Detection Delay	None	2,19 s	1,14 s	8,02 s
	EE	6,26 s	5,30 s	8,16 s
	RF	8,47 s	3,12 s	28,36 s
	EE & RF	5,38 s	4,14 s	44,15 s
Random Forest Anomaly Detection Delay	None	1,74 s	1,04 s	14,34 s
	EE	5,51 s	4,30 s	8,49 s
	RF	11,47 s	2,91 s	35,10 s
	EE & RF	4,68 s	3,75 s	45,14 s

Regarding the anomaly detection delay results from each plug-in, it was expected that the delay values were lower when no plug-in was training. However, unexpectedly, the average delay when training both plug-in simultaneously was lower than training each plug-in individually. It is suspected that the system and room temperature influenced these last results. The conclusion we obtained from the maximum anomaly detection delay values, is that training both plug-ins simultaneously may sometimes affect heavily the anomaly detection delay, but in average the system will maintain its performance.

5.1.2 Plug-ins Implementation Evaluation

The next evaluation performed were related to the Machine Learning model training duration. This comparison was performed between training duration when creating the plug-ins and train-

ing duration when the plug-ins are implemented in the STAKE system. As we can observe from the results in the Table 5.2, the training duration from the Machine Learning models when implemented as plug-ins in the STAKE system were longer. The biggest difference was noticed when training the Random Forest plug-in. The Random Forest model training duration increased more than double, while the Elliptic Envelope model training duration increased only a bit more than 6 minutes.

Table 5.2: STAKE average of 10 tests plug-ins training duration evaluation.

Plug-ins being Trained	**Plug-in creation Training duration**	**STAKE Plug-in Training duration**
Elliptic Envelope	0:34:41.71	0:40:59.27
Random Forest	1:21:05.20	3:14:40.60
Elliptic Envelope and Random Forest	1:26:02.11	4:15:13.26

The big contrast between the models training durations was unexpected. The suspected reason for this behaviour is that the values were strongly influenced by the model complexity, system temperature and room temperature. However, we believe that the training duration would not deviate significantly from the obtained results even in optimal temperature conditions.

5.1.3 Discussion

The STAKE system demonstrated being efficient in multi-threading the different processes. Additionally, we considered that the delay between the moment that a packet is captured and the moment that an anomaly is detected, and alerted, is adequate for an intrusion detection system. However, plug-in model training durations were heavily affected when multiple tasks were executed simultaneously.

The most noticeable challenge was RAM memory efficiency. If multiple plug-ins anomaly detection are activated and multiple plug-ins training are simultaneously executed, the RAM memory was heavily affected and sometimes ran out of memory. Thus, it is recommended to not train more than 2 plug-ins simultaneously.

5.2 Elliptic Envelope Plug-in

Before model optimization, data engineering was required. Since Elliptic Envelope does not rely on linear assumptions, it was concluded that correlation features would not cause issues [CC18]. Additionally, since we had an Unsupervised Learning approach with this model, data cleaning was not performed. The next data engineering consideration was feature selection. The Elliptic Envelope model is known to not work efficiently with continuous data. Therefore, the timestamp

feature was excluded for this model.

Subsequently, data transformation was considered. Given that the Elliptic Envelope model assumes that the training data is Gaussian, the training data had to be transformed into Gaussian values and normalized before training the model. Since categorical data cannot be transformed directly as Gaussian, all categorical features were label encoded beforehand. Label encoding consists in transforming categorical features in numerical values between 0 and $n\_classes - 1$, where $n\_classes$ is the number of distinct labels.

5.2.1 Model Hyper-parameters Exploration

The hyper-parameters for the Elliptic Envelope model are: *store_precision*, *assume_centered*, *support_fraction*, *random_state* and *contamination* . The selected hyper-parameters with default values were: The *store_precision* parameter, which specifies if the estimated precision is stored (default=True). The *assume_centered* parameter, which, when set to "true", the support of robust location and covariance estimates is computed, and a covariance estimate is recomputed from it, without centering the data. On the other hand, if the *assume_centered* is set to "false" (default), the robust location and covariance are directly computed with the FastMCD algorithm without additional treatment; The *support_fraction* parameter, which indicates the proportion of points to be included in the support of the raw Minimum Covariance Determinant (MCD) estimate. The MCD method is a highly robust estimator of multivariate location and scatter, for which a fast algorithm is available (FastMCD). If "None"(default), the minimum value of *support_fraction* will be used within the algorithm; At last, the *random_state* parameter, which determines the pseudo random number generator for shuffling the data.

The main hyper-parameter of this model, which was selected for optimization, is the *contamination* percentage [HRP$^+$16]. The *contamination* parameter indicates the percentage of anomalies found in the training data. Others hyper-parameters were not considered essential, hence, the default values for these hyper-parameters were used. Intuitively, it made sense to use the *contamination* percentage equal to the amount of anomaly samples in the data. The first test was performed using 43581 samples, 20% *contamination* percentage and 20% anomaly percentage, which resulted in 60,1% accuracy. The first intuition to improve the model accuracy was to lower the hyper-parameter *contamination* percentage by 5% or increase the anomaly percentage of the samples. By increasing the anomaly percentage by 5%, the algorithm improved its accuracy to 69,3%. Therefore, it was considered a 5% margin for the *contamination* hyper-parameter percentage in relation to the real anomaly data percentage.

5.2.2 Initial Sampling Size Evaluation

Before further optimizing the hyper-parameter, it was decided to first analyse the model behaviour according to the amount of samples. Different amount of samples were used to test the model behaviour, ranging from 10008 to 62381. The n samples used for training, were the n last captured packets from the dataset. It must also be considered, that dataset splitting is not performed for Unsupervised Learning plug-in models. For that reason, all data samples were used to train the model.

From the obtained results in the Table 5.3, we could state that using samples between 31155 and 43581 was the best option. However, we must also consider that the 18758 showed a more realistic behaviour in the relation between the *contamination* hyper-parameter percentage and anomaly percentage of the samples. Therefore, it was decided to evaluate the model behaviour in regard to the *contamination* hyper-parameter percentage and anomaly percentage of the samples with 18758, 31155 and 43581 samples.

Table 5.3: Elliptic Envelope behaviour in relation to data samples. TPR = True Positive Rate (Recall) and FPR = False Positive Rate.

Test ID	***Contamination***	**Samples**	**Anomaly %**	**Accuracy**	**TPR**	**FPR**	**Processing Time**
EE_1	20%	5550	25%	61,1%	77,4%	87,7%	0:10:45,232
EE_2		10008	20%	62,6%	76,6%	93,5%	0:21:30,343
EE_3			25%	65,6%	82,6%	85,3%	0:21:51,564
EE_4		18758	20%	62,6%	72%	89,7%	0:38:41,695
EE_5			25%	60,2%	76,6%	93,5%	0:34:41,713
EE_7		31155	20%	63,1%	76,9%	92,3%	1:10:13,098
EE_8			25%	67,2%	81,5%	75,6%	1:08:11,522
EE_9		43581	20%	60,1%	75,2%	99,9%	1:38:32,654
EE_10			25%	69,3%	82,9%	71,3%	1:36:50,534
EE_11		52381	25%	55,6%	73,70%	98,9%	2:10:45,435
EE_12		62381	25%	61,4%	77,6%	87,2%	3:03:59,837

5.2.3 Hyper-parameter and Anomaly Percentage Optimization

Each number of samples used for the *contamination* hyper-parameter percentage and anomaly percentage of the samples model behaviour evaluation are represented in the Tables 5.4, 5.5 and 5.6. Anomaly percentages higher than 50% was not considered, because the objective of the system is to detect anomalies in the devices behaviour and not detect the specific captured anomalies used in the model training.

The model demonstrated better results when using lower percentage of anomalies and *contamination* hyper-parameter percentage. This behaviour is logical because, the lower the *con-*

Table 5.4: Elliptic Envelope behaviour in relation to *contamination* hyper-parameter, with 18758 samples.

Samples	***Contamination***	**Anomaly %**	**Accuracy**	**TPR**	**FPR**	**Processing Time**
18758	5%	5%	90,1%	94,8%	99%	0:38:15,324
		10%	85%	94,5%	99,8%	0:40:01,432
	10%	5%	86%	90%	90,3%	0:39:05,324
		10%	81,3%	89,3%	91,5%	0:36:43,654
		15%	81,9%	90,1%	92,1%	0:36:50,435
	15%	10%	76,2%	83,7%	91,5%	0:37:20,345
		15%	71,4%	83,2%	95,5%	0:38:34,532
		20%	67,1%	82,6%	94,8%	0:37:59,435
	20%	20%	62,6%	72%	89,7%	0:38:41,695
		25%	60,2%	76,6%	93,5%	0:34:41,713

Table 5.5: Elliptic Envelope behaviour in relation to *contamination* hyper-parameter, with 31155 samples. TPR = True Positive Rate (Recall) and FPR = False Positive Rate.

Samples	***Contamination***	**Anomaly %**	**Accuracy**	**TPR**	**FPR**	**Processing Time**
31155	5%	10%	85%	94,4%	100%	1:09:45,897
		5%	90%	94,7%	100%	1:11:24,659
	10%	5%	85%	89,5%	100%	1:12:13,043
		10%	82,6%	90,4%	86,8%	1:10:43,098
		15%	75%	88,2%	100%	1:06:54,623
	15%	10%	75,9%	83,82%	95,6%	1:08:34,790
		15%	70%	82,4%	100%	1:08:10,932
		20%	67,8%	83%	93,1%	1:07:22,121
	20%	20%	63,1%	76,9%	92,3%	1:10:13,098
		25%	67,2%	81,5%	75,6%	1:08:11,522

Table 5.6: Elliptic Envelope behaviour in relation to *contamination* hyper-parameter, with 43581 samples. TPR = True Positive Rate (Recall) and FPR = False Positive Rate.

Samples	***Contamination***	**Anomaly %**	**Accuracy**	**TPR**	**FPR**	**Processing Time**
43581	5%	5%	91%	95,77%	100%	1:37:03,454
		10%	85%	94,4%	100%	1:35:13,543
	10%	5%	85%	89,5%	100%	1:36:52,129
		10%	80%	88,9%	100%	1:34:53,644
		15%	75%	88,2%	100%	1:37:43,654
	15%	10%	75%	83,3%	100%	1:40:12,324
		15%	72,2%	83,7%	92,6%	1:38:42,235
		20%	66,2%	82%	96,9%	1:37:21,123
	20%	20%	60,1%	75,2%	99,9%	1:38:32,654
		25%	69,33%	82,9%	71,3%	1:36:59,435

tamination hyper-parameter percentage, the smaller the generated ellipse will be. Thus, when high *contamination* hyper-parameter percentages are used, the risk of having high amount of benign packets identified as anomalies will be increasingly higher.

The tests using 5% *contamination* percentage with 18758, 31155 and 43581 samples showed good accuracy results. However, these tests demonstrated excessively high false positive rate. It was concluded that further model optimization was required. Therefore, it was decided to evaluate the model again according to the different number of samples, but this time using 5% *contamination* hyper-parameter percentage.

5.2.4 Optimized Hyper-parameter Sampling Size Evaluation

The initial observation we were able obtain from the results in Table 5.7 is that the model has a tendency of having high false positive rates. High false positive rates are common in Unsupervised Learning algorithms [DD11]. However, false positive rate should be minimized, since if it is excessively high, the administrator is forced to investigate each packet, which removes the point of the system.

Table 5.7: Elliptic Envelope behaviour in relation to data samples, with 5% hyper-parameter *contamination* percentage. TPR = True Positive Rate (Recall) and FPR = False Positive Rate.

Test ID	***Contamination***	**Samples**	**Anomaly %**	**Accuracy**	**TPR**	**FPR**	**Processing Time**
EE_13	5%	10008	5%	90%	94,7%	100%	0:20:22,426
EE_14		18758	5%	90,1%	94,8%	99%	0:38:15,324
EE_16		20000	5%	90,3%	95%	100%	0:40:11,472
EE_17		31155	5%	90%	94,7%	100%	1:11:24,659
EE_18		43581	5%	91%	95,77%	100%	1:37:03,454
EE_19		52381	5%	90,8%	95,62%	100%	2:08:30,755
EE_20			10%	85%	94,4%	100%	2:12:15,910
EE_21		62381	5%	93,1%	96,2%	64,9%	3:00:30,409
EE_22			10%	85%	94,5%	100%	3:05:11,432
EE_23		74881	5%	89,9%	94,6%	100%	4:10:04,898
EE_24		84488	5%	90,3%	95,1%	100%	5:05:03,423

5.2.5 Extra Optimization

The first considered solution was to reduce the number of used features. It was suspected that, since it is was used 33 features in total, the Elliptic Envelope algorithm was struggling in generating an efficient hyper-ellipsoid for anomaly detection. However, reducing the number of features has the risk of excluding the possibility of detecting anomalies from those specific removed features. Therefore, it was considered the use of PCA (Principal Component Analysis) or LDA (Linear Discriminant Analysis) as alternative. PCA is an unsupervised dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set [CC18]. On the other hand, LDA is a supervised dimensionality-reduction method. The LDA

method assumes that all classes are linearly separable, and draws hyperplanes and projects the data onto these hyperplanes in such a way as to maximize the separation of the classes [TGIH17].

Our goal of the Elliptic Envelope plug-in was to have an entirely unsupervised approach. Therefore, LDA was excluded as an option. By using PCA to reduce the training data dimensionality, we expected to not lose any information from all the features selected for anomaly detection.

The use of PCA to transform the training data into 2 dimensions demonstrated benefits in reducing the model training duration (due to smaller training data dimension), maintaining the results reproducibility and reducing the high positive rate between 5%-15%, as represented in the Table 5.8. The Elliptic Envelope algorithm has the inherent problem of result inconsistency when using data with more than 2 features (2 dimensions), which means that the final results may not be always reproducible. However, when transforming the data with PCA into 2 dimensions, the results became reproducible. Reproducibility with respect to machine learning means that you can repeatedly run your algorithm on certain datasets and obtain the same results. Moreover, reproducibility creates trust and credibility in the ML model.

Table 5.8: Elliptic Envelope behaviour in relation to data samples, using PCA and with 5% hyper-parameter *contamination* percentage. TPR = True Positive Rate (Recall) and FPR = False Positive Rate.

Test ID	***Contamination***	**Anomaly %**	**Samples**	**Accuracy**	**TPR**	**FPR**	**Processing Time**
EE_25	5%	5%	10008	91,5%	95,5%	85,4%	0:19:02.315
EE_26			18758	91,3%	95,4%	87,1%	0:37:40.532
EE_27			20000	91,3%	95,4%	87,4%	0:39:08,029
EE_28			31155	91,3%	95,4%	87,3%	1:08:53.612
EE_29			43581	91,3%	95,4%	87%	1:35:53,644
EE_30			52381	92,8%	96,2%	72,4%	2:06:45,423
EE_31			62381	93,9%	96,8%	60,8%	2:58:42,279
EE_32			74881	90%	94,8%	99,7%	4:02:32,532
EE_33			84488	90%	94,8%	99,7%	5:01:24,897

An additional PCA benefit, is that the PCA simplifies the representation (figure 5.1) of the ellipse which the model generates for anomaly detection. The representation of the generated hyper-ellipsoid was not previously possible because of the data high dimensionality.

From the plots from the figure 5.1, we were able observe that the difference between benign and anomaly are too ambiguous for the model to behave efficiently. The benign and anomalous samples are too similar for the algorithm to generate an efficient ellipse. Therefore, it was considered that the Elliptic Envelope algorithm is not able to fit well in this specific environment.

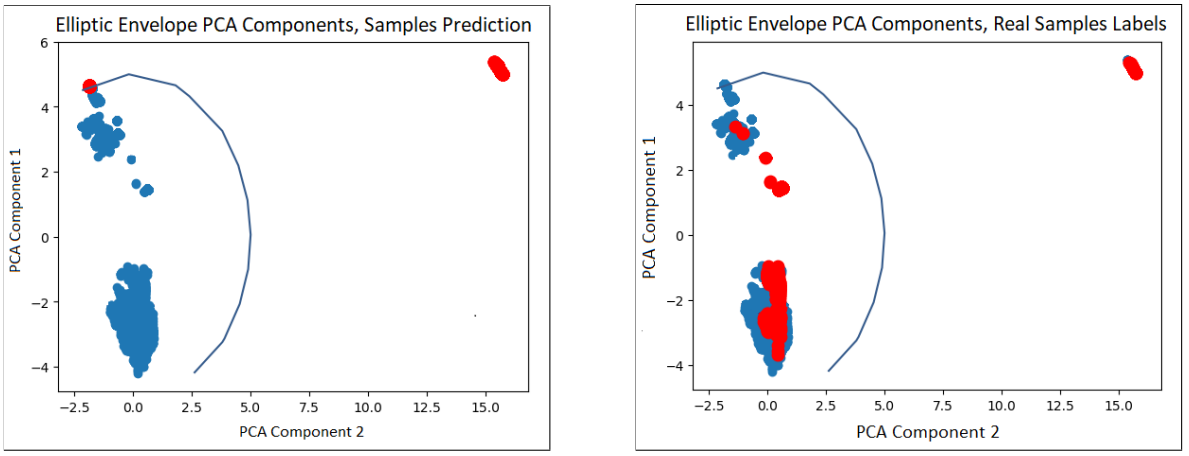

Figure 5.1: Elliptic Envelope ellipse generation representation with 43581 samples, 5% *contamination* hyper-parameter and 5% anomaly percentage, using PCA 2 dimensional transformation. The blue dots represent benign samples and the red dots represent anomalous samples.

5.2.6 Discussion

Since it was considered that no further improvement was possible, we selected the test ID 'EE_31' configuration, from the Table 5.7, for the plug-in creation. The configuration selection criteria was that this test configuration obtained the best accuracy (93,9%) with the lowest possible false positive rate (60,8%).

Due to the high false positive rate, we suspect that the model will affect the system negatively. The difference between benign and anomaly will be less clear over time. Thus, the model performance will deprecate significantly on the following trainings and will produce several false positive alerts.

We initially suspected that the gaussian data transformation that was used, may have affected negatively the final outcome. The data was transformed into gaussian values, this is a correct procedure because the model inherently expects gaussian data for training. However, by transforming into gaussian, we suspect that the difference between the benign and anomaly packets may have faded. Hence, the model derived high positive rates. Removing the gaussian transformation was not an option since the Elliptic Envelope model expects a gaussian distributed dataset.

It was also considered that the high false positive rate issue could have caused by the model difficulty in generating an efficient hyper-ellipsoid for the given features and the data transformation process. At first, we considered that the model would benefit in having multiple features to analyse from. However, we came to the conclusion that increasing the complexity for the Elliptic Envelope deprecates the model efficiency in detecting anomalies. Reducing the amount of features was not considered an option, because it will remove the model possibility

of detecting anomalies in specific protocols flags. Therefore, it was considered the use of PCA to transform the data into 2 dimensions, which improved the final results and maintained most of the information available from all the selected features.

5.3 Random Forest Plug-in

Data engineering is an essential step to perform before optimizing a Supervised Learning model. In general highly correlated features causes a problem in Random Forest models [DD11]. However, after testing the model behaviour with and without highly correlated features, no difference in the model behaviour was noticed. Since keeping the highly correlated features would not change the model classification, it was decided to keep the highly correlated features because it gives an opportunity to detect anomalies that may not be detected in other features.

As mentioned previously, the dataset featurizes each of the protocols. However, this come at the cost of having missing values. For example, UDP features will be empty in case of featurizing a TCP packet. For that reason, deleting instances with missing data was not an option.

Another consideration is the data balancing. Scaling is done to normalize data so that priority is not given to a particular feature. The objective of scaling is mostly important in algorithms that are distance based. Random Forest is a tree-based model, which uses information gain/gini coefficient inherently, hence does not require feature scaling [CC18].

The next data engineering consideration was feature selection. Similar to the Elliptic Envelope model, the Random Forest is known to not work efficiently with continuous data. Continuous data makes the Random Forest algorithm generalize poorly, as a consequence, it overfits the model. Therefore, the timestamp feature will also be excluded for this model.

The Random Forest model requires the training data to be numerical. To convert the categorical data to numerical, two options were found: One Hot Encoder and Label Encoder. As mentioned before, label encoding consists in transforming categorical features in numerical values between 0 and $n\_classes - 1$. And, on the other hand, one hot encoding consists in performing "binarization" of the categoric data and split it into multiple features. However, for a model to predict near real-time captured packets, a static number of feature dimension will be required. If the features used for prediction are different from the features used for training, an exception will be returned by the model. Therefore, One Hot Encoder will be excluded as an option for data transformation.

5.3.1 Model Hyper-parameters Exploration

Before analysing the model behaviour according to the amount of samples used, and before comparing both data transformation methods, the model hyper-parameters should be explored beforehand. The hyper-parameters available for the Random Forest model are: *n_estimators*, *criterion*, *max_depth*, *min_samples_split*, *min_samples_leaf*, *min_weight_fraction_leaf*, *max_features*, *min_impurity_decrease*, *min_impurity_split*, *bootstrap*, *oob_score*, *n_jobs*, *random_state*, *warm_start*, *class_weight*, *ccp_alpha* and *max_samples*. Taking into account the extensive set of hyper-parameters, hyper-parameter selection was recommended. The hyper-parameter selection criteria was based on parameter importance and how much it influences the model.

The selected hyper-parameters with default values are: The *criterion* parameter, which specifies the function to measure the quality of a split. Supported functions to measure the quality of split are gini impurity (default) and the information gain (entropy); The *min_weight_fraction_leaf*, which indicates the minimum weighted fraction of the sum total of weights required to be at a leaf node, the default is 0; The Random Forest model grows trees with *max_leaf_nodes* in best-first fashion. Best nodes are defined as relative reduction in impurity and if the *max_leaf_nodes* parameter is set to "none", then unlimited number of leaf nodes are grown; A tree node will be split if this split induces a decrease of the impurity greater than or equal to the *min_impurity_decrease* parameter value. The default value of the *min_impurity_decrease* is 0; The *min_impurity_split* indicates the threshold for early stopping in tree growth, the default is "none"; The *oob_score* specifies whether to use out-of-bag samples to estimate the generalization accuracy, being "false" the default value; The *n_jobs* is the number of jobs to run in parallel, the default value is "none"; The *random_state* controls both the randomness of the bootstrapping of the samples used when building trees (if bootstrap=True) and the sampling of the features to consider when looking for the best split at each node (if bootstrap=False); The *warm_start* when set to "true", the model reuses the solution of the previous call to fit and add more estimators to the ensemble, if set to "false" (default) the model just fits a whole new forest; The *class_weight* specifies the weights associated with classes in the form $\{class\_label : weight\}$. If the dictionary is not given in the *class_weight* parameter, all classes are supposed to have weight one; The *ccp_alpha* indicates the complexity parameter used for Minimal Cost-Complexity Pruning (default=0). The sub-tree with the largest cost complexity that is smaller than *ccp_alpha* will be chosen; The *max_samples* indicates the number of samples to draw from X to train each base estimator.

The selected hyper-parameters for model optimization are: The *n_estimators*, which indicates the number of trees in the forest. Usually the higher the number of trees the better to learn the data. However, adding a lot of trees can slow down the training process considerably; The

max_depth parameter defines the max number of levels in each decision tree. The deeper the tree, the more splits it has and it captures more information about the data; The *min_samples_split* equals the minimum number of samples required to split an internal node. When we increase this parameter, each tree in the forest becomes more constrained as it has to consider more samples at each node; The *min_samples_leaf* parameter is similar to *min_samples_split*, however, this parameter describes the minimum number of samples at the leafs of each tree; The *max_features* specifies the number of features to consider when looking for the best split, it can be defined by a float or integer numerical value, a square root function or a logarithmic function; The *bootstrap* denotes the method for sampling data points (with or without replacement). If *bootstrap* is set to "false", the whole dataset is used to build each tree. If *bootstrap* is set to "true", it means that some samples will be used multiple times in a single tree.

5.3.2 Hyper-parameter Behaviour Analysis

Each of the hyper-parameters was then evaluated with hyper-parameter roc plots using a set of different values. This hyper-parameter evaluation was performed with samples ranging from 8753 to 99881 packets, all with 25% of anomalies and 33% split for test data. The n samples used for training, were the n last captured packets from the dataset. By analysing these plots and table, we were able to discover the best set of values (Table 5.9) to avoid both under and overfitting. The amount of samples that showed promising results from the plots were 43581 and 62381.

Table 5.9: Random Forest hyper-parameters evaluation in relation with amount of samples.

Samples	***n_ estimators***	***max_ features***	***max_ depth***	***min_ samples_ split***	***min_ samples_ leaf***	**Plot Analysis**
8753	[25]	[5]	[2,3,4,5]	[10,15,20]	[10,15,20,25]	Underfitting
17458	[25]	[5]	[3,4,5]	[10,15,20]	[5,10,15]	Underfitting
24958	[23]	[10]	[2,3,4,5]	[10,15,20]	[10,15,20]	Acceptable
31158	[25,30]	[10]	[2,3,4,5]	[10,15,20]	[10,15,20]	Acceptable
43581	[5,10,15,20]	[5]	[2,3,4,5,6]	[10,15,20]	[5,10,15,20]	Good
47681	[25]	[5,10]	[2,3,4,5,6]	[10,20]	[10,15,20]	Acceptable
52381	[37,40,50]	[10]	[2,3,4,5,6]	[5,10,15,20]	[10,15,20]	Acceptable
62381	[25]	[5]	[3,4,5,6]	[10,20,30]	[15,20]	Good
68481	[25]	[5]	[2,3,4,5,6]	[20]	[10,15,20]	Overfitting
74881	[15,20,25,30]	[5,6,7]	[2,3,4,5]	[10,15,20]	[10,15,20,25]	Overfitting
99881	[25]	[5]	[3,4,5,6]	[20]	[20]	Overfitting

The range of values inside the dark green dashed boxes from each hyper-parameters evaluation plots (figures 5.2, 5.3 and 5.4) represents the chosen approach of the ideal range selection for model complexity. This range of values were later used for the grid search, with the objective to

optimize the model and possibly find the best hyper-parameter value combination while avoiding underfitting and overfitting. The only static hyper-parameter from the grid is *bootstrap*, this parameter is set as "false" because we desire to use all the samples in the data and avoid missing detecting any anomaly.

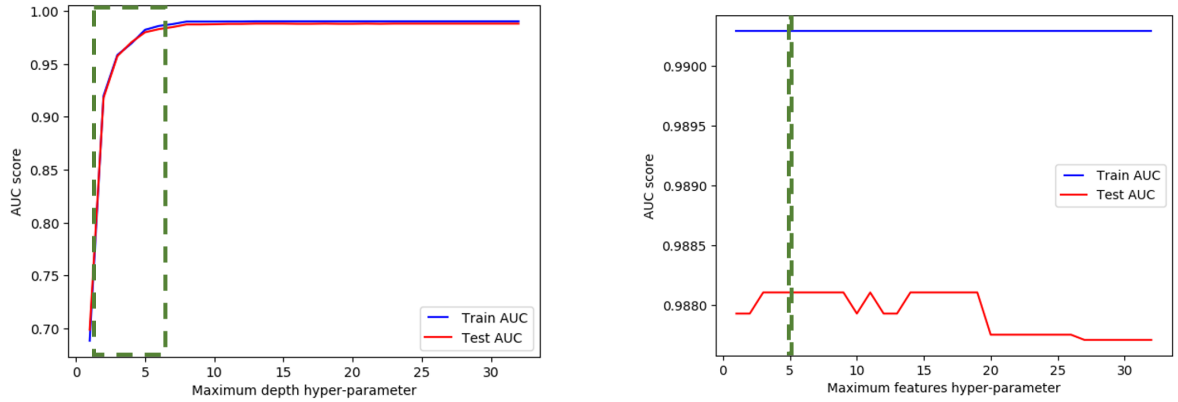

Figure 5.2: Random Forest *max_depth* and *max_features* hyper-parameters evaluation with 43581 samples.

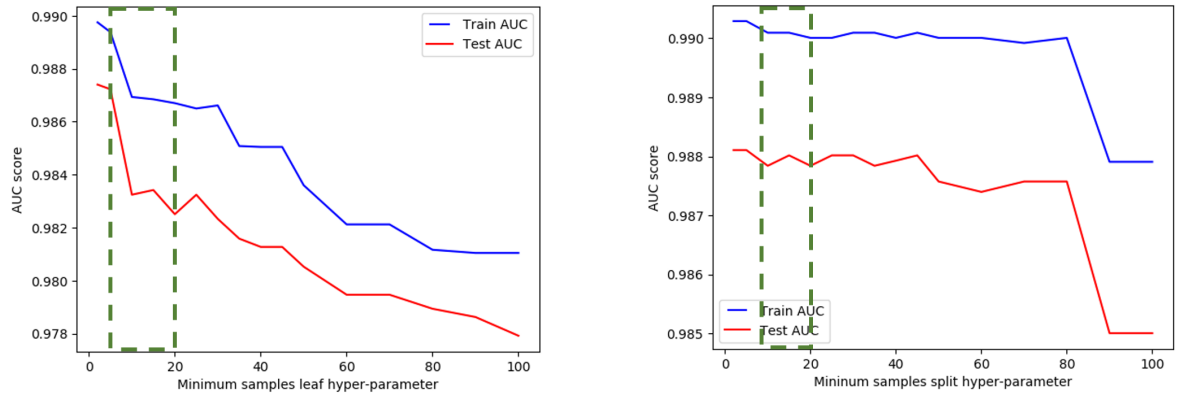

Figure 5.3: Random Forest *min_samples_leaf* and *min_samples_split* hyper-parameters evaluation with 43581 samples.

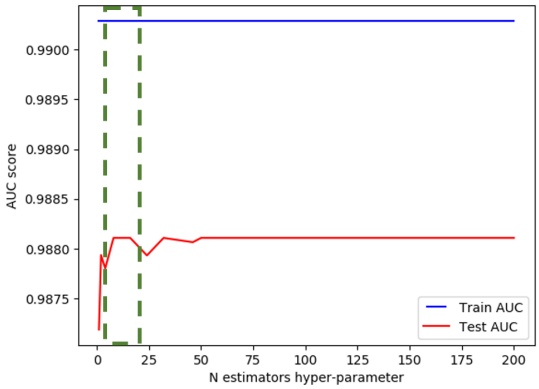

Figure 5.4: Random Forest *n_estimators* hyper-parameter evaluation with 43581 samples.

5.3.3 Anomaly Percentage Optimization

The selected amount of samples for grid search and anomaly percentage analysis was 43581 and 62381, using 33% split for test data and 5 fold cross validation. Similar to the Elliptic Envelope model, anomaly percentage higher than 50% was not considered because the objective of the system is to detect all types of anomalies in the devices behaviour and not detect the specific captured anomalies used in the training data. However, every results did overfit using grid search, achieving accuracies over 99%. In general, every Machine Learning algorithm with high complexity is inclined to overfit. It is noticeable that the Random Forest model overfits for large depth values. The trees perfectly predicts all of the train data, however, it fails to generalize the findings for new data. Therefore, it is required to control the complexity of the trees in the forest, or even prune when they grow too much. Thus, from the grid search results, it was decided to use static parameters to avoid overfitting and generalize the findings for new data as much as possible. The chosen hyper-parametrization for the Random Forest were: *bootstrap*=False, *max_depth*=2, *max_features*=10, *min_samples_leaf*=10, *min_samples_split*=20 and *n_estimators*=20.

An additional observation from the results with static hyper-parametrization in Table 5.10, is that tests ID 'RF_7', 'RF_8', 'RF_9', 'RF_10' and 'RF_11' had the highest accuracy (from ∼98% to ∼99%).

Table 5.10: Random Forest test data behaviour in relation to anomaly percentage using grid search and 43581 samples.

Test ID	Samples	Anomaly %	Accuracy	FP	FN	TN	Processing Time
RF_1	43581	10%	95,8%	597	0	789	2:27:24,659
RF_2		15%	94,1%	810	42	1079	1:54:43,297
RF_3		20%	97,5%	310	56	2151	1:39:38,940
RF_4		25%	96,7%	309	157	2727	1:23:29,983
RF_5		30%	96,7%	318	153	2999	1:21:05,196
RF_6		35%	96,7%	315	156	3404	1:24:07,411
RF_7		40%	97,7%	176	151	3918	1:24:51,811
RF_8		45%	97,8%	167	145	4341	1:19:47,581
RF_9	62381	25%	97,6%	419	74	3983	2:30:28,313
RF_10		30%	98,1%	330	54	4438	2:27:51,081
RF_11		35%	98,5%	263	48	5078	2:29:21,923

However, these tests showed hyper-parametrization overfitting in the hyper-parametrization evaluation plots (fig 5.5). Therefore, tests ID 'RF_7', 'RF_8', 'RF_9', 'RF_10' and 'RF_11' were excluded for Random Forest plug-in creation.

Furthermore, tests ID showed that 'RF_1', 'RF_2' and 'RF_3' underfitted the model. This underfitting was observable both on the evaluation scoring metrics and hyper-parametrization

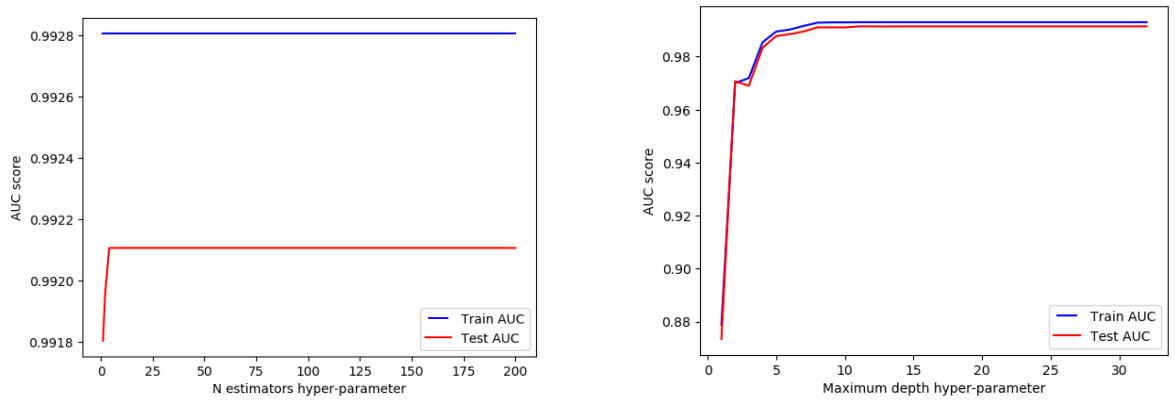

Figure 5.5: Random Forest 'RF_9' *n_estimators* overfitting and 'RF_8' *max_depth* overfitting.

evaluation plots (fig 5.6). Therefore, they were also excluded.

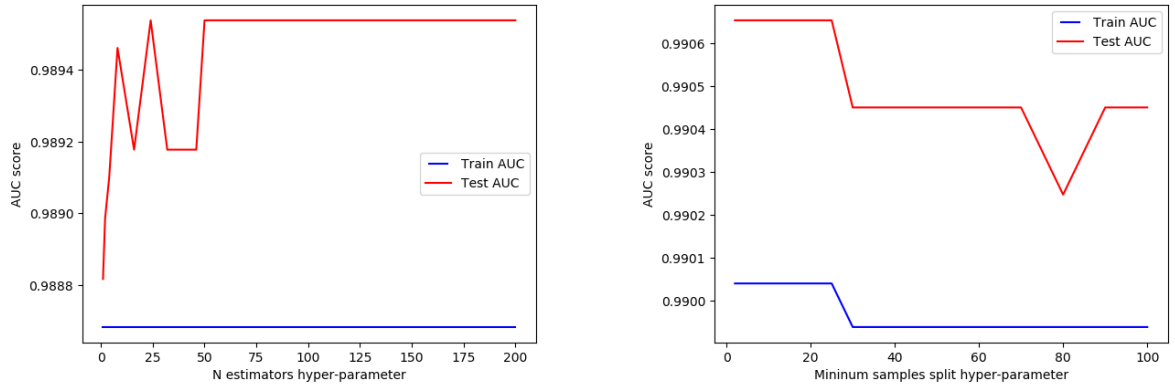

Figure 5.6: Random Forest 'RF_2' *n_estimators* underfitting and 'RF_3' *min_samples_splits* underfitting.

The remaining tests are 'RF_4', 'RF_5' and 'RF_6'. Since all of them showed similar results, further score evaluation was required. However, all of the tests scoring results still showed similar values (Table 5.11). Even though all these previous model configurations could be used for the Random Forest plug-in creation, test ID 'RF_5' configuration was selected because it has a middle ground anomaly percentage, which theoretically increases the probability of data flexibility in case the network changes its behaviour.

Table 5.11: Random Forest best tests results comparison.

Test ID	**Recall**	**Precision**	**F-score**	**Specificity**	**False Positive Rate**	**MSE**
RF_4	98,6%	97,4%	98%	89,1%	10,9%	0,1296
RF_5	98,6%	97,2%	97,9%	90,4%	9,6%	0,131
RF_6	98,5%	97,1%	97,8%	91,5%	8,5%	0,131

5.3.4 Extra Optimization

A further consideration was feature importance. Feature importance assigns a score to input features based on how useful they are at predicting a target variable. The trained model from test ID 'RF_5' assigned 0 importance to 16 features. A good practice would be removing these 0 coefficient value features. Nevertheless, the model behaviour and evaluation metrics scores did not change by removing these features. In addition, since a Smart Home network may change its behaviour pattern, its difficult to know if these features may be useful in the future. Therefore, no action will be taken about the 0 importance valued features.

Last consideration taken was Random Forest result consistency. On the contrary from Elliptic Envelope, Random Forest results are reproducible no matter the amount of features used. Which means that the model has the tendency to obtain the same result with the same model and data configuration.

5.3.5 Discussion

The results of the Random Forest model showed potential in anomaly detection efficiency. The biggest perceived challenge with this model was to avoid overfitting. However, this was expected because it is a common problem with this model. For this model to work the most efficiently possible, there must be a clear dissimilarity between benign and anomalous behaviour. This implies that the Random Forest plug-in will work very efficiently if the devices behaviour does not change frequently and if the dataset is not heavily affected by false positive and/or false negatives.

An additional consideration is that implementing other type of models as plug-ins in parallel with the Random Forest plug-in should be done with care. Since an inefficient plug-in could affect heavily the Random Forest plug-in performance.

5.4 Plug-ins Re-training Evaluation

The last evaluation was the plug-ins training result comparison between plug-in creation results and plug-in re-trained in STAKE system results. Since the system is capturing in real-time and changes in the data are expected, the only difference between these two situations will be the data used for training. The data used to train each plug-in in the STAKE system was: the original training data, plus one day of capture while the plug-in anomaly detection was active. This testing was performed individually, for each plug-in, with the objective to avoid that one plug-in would affect the other final results. All the other procedures performed and

configurations sets used were exactly the same as the ones used during plug-in creation.

As we can observe from the Tables 5.12 and 5.13, both Elliptic Envelope and Random Forest plug-in training results worsen when re-trained in STAKE system. Even though it was expected for the Elliptic Envelope to have inconsistent results with data changes, it was unexpected for the Random Forest to overfit because training process was exactly the same and this model is known to obtain consistent results. It was evident that the plug-ins were overly optimized for the specific trained data and not configured for flexibility. Hence, with changes in the training data, the Random Forest started to overfit. Thus, further plug-in testing was performed while implemented in STAKE system.

Table 5.12: Elliptic Envelope plug-in creation training and STAKE implementation training results comparison.

Test ID	**Accuracy**	**Recall**	**Precision**	**F-score**	**Specificity**	**False Positive Rate**	**Mahalanobis [Mininum; Maximum]**
EE_31	93,9%	96,8%	96,8%	96,8%	39,2%	60,8%	[0,0006; 7350,1820]
EE_ST	90%	94,7%	94,7%	94,7%	0%	100%	[0.0002; 1359,8267]

Table 5.13: Random Forest plug-in creation training and STAKE implementation training results comparison.

Test ID	**Accuracy**	**Recall**	**Precision**	**F-score**	**Specificity**	**False Positive Rate**	**MSE**
RF_5	96,7%	98,6%	97,4%	98%	89,1%	10,9%	0,1296
RF_ST	100%	100%	100%	100%	100%	0%	0,0

As a result of the high false positive rate of the Elliptic Envelope model, the comparison between anomaly and benign packets became less clear for the model over time. Therefore, the model scoring has worsen when re-trained in the STAKE system. Alternative number of samples and alternative percentage values for the *contamination* hyper-parameter were tested, however, same problems were presented.

Regarding the Random Forest plug-in, further evaluation was performed by testing other alternative configurations and different anomaly percentages. These alternative configurations involved in limiting even further the hyper-parameter optimization (Table 5.14) with the objective of avoiding overfitting as much as possible. Regarding the set of anomaly percentages used, it consisted of 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40% and 45%.

However, using both 43581 and 62381 samples, all configurations settings still overfitted with the same results as test ID 'RF_ST' in Table 5.13. Further tests were executed by decreasing

Table 5.14: Random Forest hyper-parameters evaluation in relation with amount of samples.

Plug-in ID	***n_ estimators***	***max_ features***	***max_ depth***	***min_ samples_ split***	***min_ samples_ leaf***
RF_SL_1	20	10	2	20	10
RF_SL_2	20	10	1	20	10
RF_SL_3	3	2	1	80	60

the number of features gradually. Nevertheless, the model behaviour did not change from this action and still overfitted. Plus, removing some features will make the system insecure for some attacks.

5.4.1 Discussion

Regarding the developed plug-ins, Elliptic Envelope demonstrated tendency to produce excessively high false positive rates. This behaviour affects heavily all the implemented plug-ins, because the difference between benign and anomaly packets will be less clear over time. We concluded that the Elliptic Envelope has the benefit of being flexible to changes. However, the model demonstrated serious difficulty in identifying anomalies in this specific environment. Thus, it was considered that no further optimization was possible with this plug-in in this specific environment.

On the other hand, Random Forest plug-in demonstrated an exceedingly tendency for overfitting in this kind of environment. The problem in this situation is that the model is memorizing the packets, therefore it will be hard for the model to generalize and detect new device behaviours. This will cause the system to have high amount of false negative and positives. We considered that the Random Forest is a powerful algorithm when optimized for specific static environments. However, not flexible enough for dynamic environments, which is the case of most Smart Home networks. For this plug-in to work efficiently, it will require adjustments each time it will be trained.

5.5 Summary

In this Chapter we started by discussing the STAKE system development results, by comparing the system and plug-ins performance in different situations. Afterwards, we discussed the results from both the Elliptic Envelope plug-in and the Random Forest plug-in. Finally, we discussed the difference between the plug-in training results when optimizing the models and the plug-in training results when re-trained in the system.

Chapter 6

Conclusion

This dissertation described the design, implementation, evaluation methodology and results of STAKE, a solution for capturing and storing Smart Home network traffic, and an execution environment for anomaly detection plug-ins. This Chapter presents the achievements of STAKE and describes future work that can be developed to improve STAKE.

STAKE's novel contribution is the execution environment. It was evaluated in a test-bed with data from real devices. The system, demonstrated flexibility in implementing different types of plug-ins. Even though some adjustment was required for each plug-in, this could be easily and quickly performed in the user interface. The changes in the plug-in configuration that takes longer time are model hyper-parameters related, which cannot be performed directly on user interface and requires a new trained model to be uploaded as plug-in. Nevertheless, changing model hyper-parameters is usually a frequent action when optimizing the plug-in and not when implemented in a system.

The benign and anomaly packets segregation system component exhibited the drawback in adding false positives in the benign dataset and false negatives in anomaly dataset. However, this component showed benefits in simplifying anomaly analysis in different detail levels, simplifying the discovery of the source of anomaly and identifying the false positives and negatives. Additionally, with the support of device manager component, if the devices were previously correctly identified by the administrator, the identification of the devices affected by the anomalies are easily identified by the system and easily evaluated in the segregated datasets.

In addition, the hardware used for evaluation demonstrated being efficient in threading the local HTTP server hosting, data collection (packet capturing), plug-ins training and plug-ins anomaly detection processes. The hardware evaluation results shows that the CPU is able to thread all the processes, along with multiple plug-ins, without being strongly affected by it. The most affected hardware element, even though moderately, was the RAM memory. However, this

was expected because each plug-in needs to store the training data individually. Even though multiple plug-ins did not strongly influence the overall anomaly detection delay, it still strongly affected the plug-ins training duration. From these observations, we can conclude that the selected hardware supports the system and multiple plug-ins efficiently.

Transfer learning can be performed in the STAKE system. This process consists of taking a model trained on one dataset and then use it to predict another related problem. However, this method might perform well in some plug-ins, but will not in many [CC18]. Therefore, for the plug-ins to be effective, it is recommended to re-train each plug-in after a period of time.

If the features used for prediction are different from the features used for training, an exception will be returned by the model. This is a normal behaviour from machine learning models, however, should be taken into consideration when implementing a new plug-in.

Regarding the results of the plug-ins when being created and optimized, the Elliptic Envelope obtained reasonable results, while Random Forest obtained good results. Nevertheless, more can be developed and/or improved, because this work will be available to the community at large. However, on the contrary from the plug-in creation results, when the plug-ins were re-trained in the system, each of the plug-ins behaviour and results deteriorated. As regard to the Elliptic Envelope plug-in, the model demonstrated tendency to produce excessively high false positive rates. On the other hand, Random Forest plug-in demonstrated an exceedingly tendency for overfitting in this kind of environment.

6.1 Achievements

The STAKE system user interface is accessible via HTTP. After the administrator is authenticated, the administrator is able to: edit his/her account information, identify the network devices according to the associated mac address and add/edit/remove plug-ins. Additionally, the administrator has access to information such as: captured packets statistics, system hardware status information (hard disk memory usage, cpu usage, ram memory usage and network bandwidth usage), system log, anomalies alerts, plug-in status and plug-in training results.

In regard to plug-in development, both Elliptic Envelope and Random Forest based plug-ins were successfully created. Both plug-ins demonstrated promising results when they were created. However, when re-trained in the system the plug-ins behaviour deteriorated. This is a common problem in environments that change frequently. Hence, for good re-training results, a plug-in with a flexible and reproducible model or integrating human feedback will need to be considered for future work. A human feedback can be used to fine tune the global comprehension of the anomalies, while preserving the efficiency of the system and plug-ins [DD11].

Even though the plug-ins implementation results were disappointing, the system showed promising results. The system demonstrated being capable of announcing anomalies in a timely manner, even when training two plug-ins simultaneously. Additionally, the system processor exhibited being capable of withstanding the threading architecture. Nevertheless, the most noticeable drawback in the system was the RAM memory limitation, which caused exceptions when a large number of samples were used. To avoid this problem, it is recommended to train, at maximum, two plug-ins simultaneously in this selected hardware.

6.2 Future Work

Regarding future work, we propose the following implementations suggestions for further system improvements. The list is ordered top-bottom from the suggestions/improvements we think are more interesting in our perspective:

1. Implement more plug-ins with different algorithms, such as clustering algorithms that could be used to find anomalies relation, origin and data flow of the intrusion. Algorithm with clustering approaches allows us to find anomalies correlation and possibly the source of the system vulnerability. Additionally, it would be also interesting to see the system behaviour with a plug-in which algorithm uses time series as training data.

2. Add the option to order plug-in priority for anomaly detection. There is an extensive set of Machine Learning models to choose from and some models will work more efficiently than others on the given Smart Home environment. Giving priority to the most efficient plug-ins would probably reduce the amount of detected false positives on the system. The level of priority could be configured by ordering or weighting the plug-ins priority.

3. Integrating human feedback should be considered. If security analysts are able to report false positives and false negatives directly to a system that adjusts model parameters based on this feedback, the maintainability and flexibility of the system can be vastly elevated. In untrusted environments, however, directly integrating human feedback into the model training can have negative effects.

4. Add more protocols for anomaly detection, such as: HTTP (Hypertext Transfer Protocol), HTTPS (Hypertext Transfer Protocol Secure), FTP (File Transfer Protocol), SSH (Secure Socket Shell), Telnet, IMAP (Internet Message Access Protocol), DNS (Domain Name System) and IRC (Internet Relay Chat). Aggregating more protocols for anomaly

detection, which are used in a Smart Home environment, could increase the probability of a more effective anomaly detection and a more detailed analysis of the source of intrusion.

5. The current system is analysing multiple protocols in the same dataset. A possible alternative approach would be to analyse each protocol individually. This alternative could make the anomaly detection of the system more efficient or affect the anomaly detection negatively.

6. The STAKE system returns alerts and a short reports about the detected anomalies. However, the report could be upgraded to a full detailed report. This full detailed report could have information such as: intrusion origin, which part of the system or devices were affected, which plug-ins detected this anomaly, possible solution, etc...

7. Blocking devices with anomalous behaviour should be an option. However, it must be taken into account that this will prevent collecting any further information about the attackers capabilities, intent and origin. Additionally, it must also be taken into account that this option could wrongly block benign devices when false negatives are detected. Therefore, this option is recommended to be implemented in conciliation with the human feedback integration feature.

Bibliography

[AM18] Aung, Yi ; Min, Myat: Hybrid Intrusion Detection System using K-means and Random Tree Algorithms. https://ieeexplore.ieee.org/abstract/document/8441094 : University of Computer Studies, Mandalay Mandalay, Myanmar, Juni 2018

[BCN18] In: Bottou, Léon ; Curtis, Frank E. ; Nocedal, Jorge: *Optimization Methods for Large-Scale Machine Learning.* https://epubs.siam.org/doi/10.1137/16M1080173, 2018

[Ber18] Berte, Dan-Radu: Defining IoT. (2018), 05

[CC18] Clarence Chio, David F.: *Machine Learning and Security.* O'Reilly Media, Inc., 2018. – ISBN 9781491979853

[CGM+19] Cramer, Irene ; Govindarajan, Prakash ; Martin, Minu ; Savinov, Alexandr ; Shekhawat, Arun ; Staerk, Alexander ; Thirugnana, Appasamy: Detecting Anomalies in Device Event Data in the IoT. https://pdfs.semanticscholar.org/7e3c/9790f6ff6fe2fcbe691ca21a5113a4c25814.pdf : Bosch Software Innovations GmbH, IoT Analytics, März 2019

[CP19] Carmo, Pedro E. ; Pardal, Miguel L.: IoT Neighborhood Watch: device monitoring for anomaly detection. http://web.tecnico.ulisboa.pt/~miguel.pardal/www/pubs/2019_Carmo_Pardal_INForum_NWatch.pdf : Instituto Superior Técnico, Universidade de Lisboa, Dezember 2019

[DAF18] Doshi, Rohan ; Apthorpe, Noah ; Feamster, Nick: Machine Learning DDoS Detection for Consumer Internet of Things Devices. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8424629 : Department of Computer Science Princeton University Princeton, New Jersey, USA, 2018

[DCL18] Drai, David ; Cohen, Ira ; Lang, Shay ; Anodot (Hrsg.): *Ultimate Guide*

To Building A Machine Learning Anomaly Detection System. https://www.kdnuggets.com/2017/05/anodot-machine-learning-anomaly-detection.html : Anodot, 2018

[DD11] Dua, Sumeet ; Du, Xian: *Data Mining and Machine Learning in Cybersecurity.* Taylor and Francis Group, LLC, 2011. – ISBN 9781439839430

[EA19] Ekholm, Albin ; Avdic, Adnan: *Anomaly Detection in an e-Transaction System using Data Driven Machine Learning Models.* http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1335216&dswid=-7153, 07 2019

[HIZH19] Hasan, Mahmudul ; Islam, Milon ; Zarif, Ishrak I. ; Hashem, M.M.A.: Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches. https://www.sciencedirect.com/science/article/pii/S2542660519300241 : Internet of Things Research Lab, Department of Computer Science and Engineering, März 2019

[Hos18] Hosmer, Chet: *Defending IoT Infrastructures with the Raspberry Pi.* Apress, 2018. – ISBN 9781484237007

[HRP+16] Hoyle, Ben ; Rau, Markus M. ; Paech, Kerstin ; Bonnett, Christopher ; Seitz, Stella ; Weller, Jochen: Anomaly detection for machine learning redshifts applied to SDSS galaxies. (2016), 06

[KDL19] Kamaraj, Kavin ; Dezfouliy, Behnam ; Liuz, Yuhong: Edge Mining on IoT Devices Using Anomaly Detection. http://www.apsipa.org/proceedings/2019/pdfs/295.pdf : Internet of Things Research Lab, Department of Computer Science and Engineering, November 2019

[LZT+19] Li, Zhen ; Zou, Deqing ; Tang, Jing ; Zhang, Zhihao ; Sun, Mingqian ; Jin, Hai: A Comparative Study of Deep Learning-Based Vulnerability Detection System. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8769937 : National Key Research and Development (R&D) Plan of China, Juli 2019

[OM19] Oliveira, Gil de ; Mouta, Varão: SPATIO: end-uSer Protection Against IoT IntrusiOns, Instituto Superior Técnico, Universidade de Lisboa, Oktober 2019

[TGIH17] Tharwat, Alaa ; Gaber, Tarek ; Ibrahim, Abdelhameed ; Hassanien, Ella: Linear discriminant analysis: A detailed tutorial. (2017), 05

[WPC19] Wang, Sheng ; Pardal, Miguel L. ; Claro, Rui: SPYKE: Security ProxY with Knowledge-based intrusion prEvention. http://web.tecnico.ulisboa.pt/~miguel.pardal/www/pubs/2019_Wang_Pardal_INForum_SPYKE.pdf : Instituto Superior Técnico, Universidade de Lisboa, Dezember 2019

[WZZ19] Wang, Haoran ; Zariphopoulou, Thaleia ; Zhou, Xunyu: Exploration versus exploitation in reinforcement learning: a stochastic control approach. (2019), 01

[XWL+18] Xiao, Liang ; Wan, Xiaoyue ; Lu, Xiaozhen ; Zhang, Yanyong ; Wu, Di: IoT Security Techniques Based on Machine Learning: How Do IoT Devices Use AI to Enhance Security? https://ieeexplore.ieee.org/abstract/document/8454402 : University of Computer Studies, Mandalay Mandalay, Myanmar, 2018

[ZACF18] Zheng, Serena ; Apthorpe, Noah ; Chetty, Marshini ; Feamster, Nick: User Perceptions of Smart Home IoT Privacy. https://dl.acm.org/doi/pdf/10.1145/3274469?download=false : Princeton University, USA, November 2018