Universidade Estadual de Londrina

VITOR HUGO BEZERRA

BOTNET DETECTION IN INTERNET OF THINGS DEVICES USING ONE-CLASS CLASSIFICATION

Londrina
2019

VITOR HUGO BEZERRA

BOTNET DETECTION IN INTERNET OF THINGS DEVICES USING ONE-CLASS CLASSIFICATION

Master's Thesis presented to the Master's Program in Computer Science of the Universidade Estadual de Londrina for the title of Master in Science in Computer Science.

Supervisor: Prof. Dr. Bruno Bogaz Zarpelão

Londrina
2019

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Bezerra, Vitor Hugo .
Botnet detection in Internet of Things devices using One-class classification / Vitor Hugo Bezerra. - Londrina, 2019.
69 f.

Orientador: Bruno Bogaz Zarpelão.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2019.
Inclui bibliografia.

1. Internet das Coisas - Tese. 2. Segurança em Redes - Tese. 3. Detecção de Anomalias - Tese. 4. IoT Dataset - Tese. I. Zarpelão, Bruno Bogaz . II. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. III. Título.

VITOR HUGO BEZERRA

BOTNET DETECTION IN INTERNET OF THINGS DEVICES USING ONE-CLASS CLASSIFICATION

Master's Thesis presented to the Master's Program in Computer Science of the Universidade Estadual de Londrina for the title of Master in Science in Computer Science.

EXAMINATION BOARD

Supervisor: Prof. Dr. Bruno Bogaz Zarpelão
Universidade Estadual de Londrina – UEL

Prof. Dr. Sylvio Barbon Jr.
Universidade Estadual de Londrina – UEL

Prof. Dr. Elieser Botelho Manhas Jr.
Universidade Estadual de Londrina – UEL

Prof. Dr. Luiz Fernando Carvalho
Universidade Tecnológica Federal do Paraná – UTFPR

Londrina, 28th de March de 2019.

ACKNOWLEDGEMENTS

Os agradecimentos principais são direcionados à minha família pelo grande suporte na minha educação tanto acadêmica como pessoal em todos esses anos.

Agradeço a todos meus amigos da graduação/mestrado que vem apoiando tanto nos momentos de alegria quanto nos de dificuldades. Aos amigos nos laboratórios Remid e Gaia por sempre ajudar em trabalhos, ideias e até mesmo a fazer um cafezinho a tarde.

Também agradeço a todos os professores durante o mestrado, em especial os professores Sylvio e Rodrigo Miami nos trabalhos sendo desenvolvidos, o professor Daniel Kaster pelo apoio no mestrado e oferecer oportunidades para enriquecer meu currículo acadêmico, e por último ao professor Bruno Zarpelão pela orientação durante o mestrado, sempre com muita paciência e apoio na pesquisa como também na vida pessoal se tornando um grande amigo.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

"Success consists of going from failure
to failure without loss of enthusiasm."
(Winston Churchill)

BEZERRA, V. H. **Detecção de botnets em dispositivos de Internet das coisas usando classificadores One-class.** 2019. 69 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2019.

RESUMO

Com a chegada de diferentes dispositivos de Internet das Coisas (Internet of Things – IoT), novas ameaças à segurança de redes surgem devido à baixa segurança desses dispositivos. Botnets constituem uma ameaça que aproveita as vulnerabilidades dos dispositivos IoT para infectar vários deles e realizar ataques coordenados de larga escala. Para apoiar a solução desse problema, novos métodos de detecção de botnets em IoT são necessários. Neste trabalho, é proposto um sistema host-based de detecção de intrusões, chamado IoTDS (Internet of Things Detection System), para dispositivos IoT utilizando classificadores one-class. Esses classificadores são um tipo particular de algoritmos de aprendizado de máquina (Machine Learning – ML) em que é modelado um único padrão para apontar se as instâncias pertencem a essa classe ou não. Dessa forma, usando este tipo de algoritmo, é possível criar um modelo para detecção de intrusões com base apenas no comportamento legítimo do dispositivo. Na abordagem proposta, foram testados quatro classificadores one-class, Elliptic Envelope, Isolation Forest, Local Outlier Factor e Oneclass Support Vector Machine sobre dados de uso de CPU e de memória, temperatura de CPU e número de tarefas em execução coletados no dispositivo para detectar as botnets. Para desenvolver e avaliar esse tipo de proposta, é necessário que se tenha à disposição conjuntos com dados sobre os comportamentos dos ataques, assim como os comportamentos legítimos dos dispositivos. Esse tipo de conjunto de dados não é normalmente disponibilizado publicamente, pois pode acabar revelando dados sensíveis dos dispositivos monitorados. Dessa forma, para testar a abordagem, foi construído um ambiente de testes para gerar um conjunto contendo dados de um dispositivo IoT que foi infectado por botnets em um ambiente controlado. Esse conjunto de dados foi gerado a partir de três diferentes perfis de dispositivos de IoT e sete botnets de IoT. Os resultados indicam que o sistema proposto utilizando um classificador one-class é eficiente na detecção de diferentes botnets e possui um baixo consumo de recursos. O conjunto de dados desenvolvido será disponibilizado publicamente e poderá ser usado para apoiar o desenvolvimento de outras ferramentas de defesa para dispositivos IoT.

Palavras-chave: Internet das Coisas. Botnet. Segurança em redes. Detecção de anomalias. IoT Dataset.

BEZERRA, V. H. **Botnet detection in Internet of Things devices using One-class classification.** 2019. 69 p. Dissertation (Master's Deegre in Science in Computer Science) – Universidade Estadual de Londrina, Londrina, 2019.

ABSTRACT

With the increasing number of different Internet of Things (IoT) devices, new threats to network security emerge due to these devices' low security. Botnets are a widespread threat that takes advantage of IoT devices vulnerabilities to compromise multiple devices and carries out coordinated and large-scale attacks. To tackle this, new methods addressing IoT botnets detection are required. In this work, we propose a host-based detection system, named IoTDS (Internet of Things Detection System), based on one-class classifiers. They are a particular kind of machine learning (ML) algorithm that models a single pattern for detecting true instances. By using this type of ML algorithm, only the legitimate device behaviour is required to be modelled for further detection of deviations. In the proposed approach, four one-class algorithms, Elliptic Envelope, Isolation Forest, Local Outlier Factor and One-class Support Vector Machine, were tested using features extracted from CPU and memory utilisation, number of running tasks and CPU temperature to detect malicious activities. To develop and evaluate this type of approach, it is necessary to have available datasets with data describing the behaviour of botnet attacks, as well as the legitimate behaviour of these devices. This type of dataset usually is not distributed publicly due to the possibility of revealing sensitive data about the monitored devices. Thus, to test our approach, an experimental setup was constructed to generate a dataset that contains data from a IoT device that was infected by botnet malware in a controlled environment. This dataset was generated using three different IoT device profiles and seven IoT botnets. The results indicate that the proposed system has a good predictive performance for different botnets and has a low impact in the device's CPU and memory utilisation, and energy consumption. The developed dataset will be available for the research community and can be used to support the development of defence tools for IoT devices.

Keywords: Internet of Things. Botnet. Network security. Anomaly detection. IoT Dataset.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ARM	Advanced RISC Machine
AUC	Area Under the Curve
BLE	Bluetooth Low Energy
C&C	Command and Control
CAIDA	Center for Applied Internet Data Analysis
CIC	Canadian Institute for Cybersecurity
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CSV	Comma-separated Values
DARPA	Defence Advanced Research Projects Agency
DDoS	Distributed Denial of Service
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DoS	Denial of Service
DSL	Digital Subscriber Line
DHT	BitTorrent Distributed Hash Table
DVWA	Damn Vulnerable Web Application
EE	Elliptic Envelope
HIDS	Host-based IDS
HFC	Hybrid Fiber Coaxial
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IF	Isolation Forest

IoT	Internet of Things
KDD	Knowledge Discovery and Data Mining
LOF	Local Outlier Factor
MC	Multimedia Centre
MIPS	Microprocessor without Interlocked Pipeline Stages
ML	Machine Learning
MQTT	Message Queue Telemetry Transport
NIDS	Network-based IDS
OSVM	One-class Support Vector Machine
PCA	Principal Component Analysis
PCAP	Packet Capture
RPL	IPv6 Routing Protocol for Low Power and Lossy Network
SC	Surveillance Camera
SSH	Secure Shell
SSL	Secure Socket Layer
ST	Surveillance Camera with Additional Traffic
SVM	Support Vector Machine
TLS	Transport Layer Security

CONTENTS

1 INTRODUCTION

Internet of Things (IoT) devices can be found in our everyday life in many situations, e.g., surveillance cameras, healthcare monitors and traffic monitoring services. The IoT paradigm makes machine-to-machine communication over the Internet more practical, connecting more devices online and allowing them to actively participate in the network [1]. IoT environments usually consist of heterogeneous and low-cost devices with little or no security embedded into them, which collect and generate a vast amount of private information, and may create many security problems. This open wound in IoT security is likely to prevail for years to come and must not be ignored given the broad range of applications of the IoT paradigm [2].

The increase in the number of deployed IoT devices has drawn the attention of malicious users, who target those devices to gather computation power and carry out illegal activities. Those users can, for example, perform Distributed Denial of Service (DDoS) attacks by creating large-scale IoT-based botnets [2]. Moreover, compromised devices may not demonstrate any apparent symptoms of infection, being able to carry on with the execution of their normal activities. Therefore, detecting compromised devices is a challenging subject and requires specialised tools [3].

One of the many threats IoT devices face is botnets. A botnet is a collection of compromised devices, referred to as bots, controlled by one or more malicious users, which communicate with the bots to perform malicious activities. IoT botnets have been making a certain impact in our lives, having the Mirai attack in 2016[1] as the most remarkable example so far. In this attack, a botnet called Mirai infected surveillance cameras - a type of IoT device - by taking advantage of their default security settings and performed a large-scale DDoS attack against Dyn, a major DNS service provider [4].

With those constant threats being developed and the lack of tools to patch IoT devices, users can not rely on updates for every security breach found. This scenario makes an intrusion detection system (IDS) fundamental for these devices [5]. Although the research community has proposed IDS for IoT devices [6, 7], they still present some issues. The first issue is that works are mostly focused on protecting wireless sensor networks based on the 6LoWPAN protocol stack and the Contiki operating system. These networks are an important part of the IoT universe, but there are many other kinds of IoT devices and networks that also deserve attention, like the domestic devices.

Relying on algorithms that require labelled data to be trained is another problem. Labelled samples of botnets attacks are difficult to obtain, and this data also usually

[1] http://money.cnn.com/2016/10/22/technology/cyberattack-dyn-ddos/index.html

becomes quickly outdated by new botnets and attacks. The next problem is the lack of datasets to aid in the understanding of botnets, as already pointed out by Garcia et al. [8]. This problem occurs because security analysts are concerned about their data privacy and then avoid sharing them. Additionally, there are only a few botnets available for studying [8]. Lastly, most of works explore only network data for detecting attacks. It may be an issue since most of the communications are protected by end-to-end encryption, making more difficult to obtain useful information from traffic data.

This work aims to propose solutions to address these issues. We propose a host-based detection system for IoT devices using one-class classifiers. One-class classifiers are a different type of machine learning (ML) technique, which instead of classifying an instance in one of multiple pre-defined patterns, they model a single pattern and use it to discern whether a new instance belongs to the pattern or not. IoT devices have a specialised behaviour, performing simple tasks with a well-defined use of computational resources. We believe that is possible to model the device's CPU and memory utilisation, number of running tasks and CPU temperature with a one-class classifier to support the detection of a botnet infection as an anomaly in the device's behaviour. The main advantage of this proposal is to build this model without needing a specialist to label the data or having access to data on compromised behaviour. In this work, four one-class classifiers were evaluated, namely Elliptic Envolope (EE), Isolation Forest (IF), Local Outlier Factor (LOF) and One-class Support Vector Machine (OSVM). This work aims to protect a more robust type of IoT devices, which usually have an operating system for constrained devices (e.g. Raspbian and Android) and at least a 512 MB RAM (Random Access Memory). These devices are found today in many homes and offices being used in diverse applications. Examples of this type of device are the Raspberry Pi[2], Odroid[3], Google's Chromecast[4] and Roku TV[5].

To test our approach, we developed an experimental setup to generate a dataset containing data collected from a IoT device, a Raspberry Pi, which was infected by botnet malware. The collected data includes legitimate and malicious behaviour. Legitimate data concerns the emulation of regular behaviour of domestic IoT devices, such as surveillance cameras and media centres. On the other hand, the malicious data is related to botnets activities, including communication with the Command & Control (C&C) and some DDoS attacks. In our tests, the proposed approach presented a great performance in detecting botnets, considering three different device profiles and seven botnets, whilst it had a low impact in the utilisation of CPU and memory and in the consumption of energy.

The main contributions of this work can be summarised as follows:

[2] https://www.raspberrypi.org/
[3] https://www.hardkernel.com/
[4] https://store.google.com/product/chromecast
[5] https://www.roku.com/en-gb/

- Proposing a system that employs one-class classifiers to detect IoT botnets using only data about the device resources consumption, having a high predictive performance without interfering with the devices' functionality;
- Creating a controlled environment that has behaviours and functions of domestic IoT devices in a network and can be infected by IoT botnet malware.

This work is organised as follows: Chapter 2 presents the theoretical foundation needed for the comprehension of the work. In Chapter 3 the description of the proposed host-based detection approach is presented. Chapter 4 describes the experimental setup constructed. In Chapter 5 we present the results and the discussion about them. Lastly, Chapter 6 provides the conclusion of this work.

2 THEORETICAL BACKGROUND

This chapter provides information about the concepts used in this work, describing the Internet of Things paradigm, the botnets attached to it, the systems made to protect IoT systems and the methods used by these systems.

2.1 Internet of Things

The Internet of Things (IoT) is a novel paradigm that is quickly growing in the scenario of modern wireless and machine-to-machine telecommunications. This paradigm is composed by low-cost devices with limited computing resources, e.g. surveillance cameras, temperature sensors and baby monitors, which are responsible for specific tasks that do not require much computational capacity [2], and are connected through wireless networks [9, 10].

IoT devices come from a large range of different computational power and specifications [11], varying from tiny devices like the CM5000 TelosB mote module[1] that has a 10 KB RAM and a 48 KB flash memory to more robust ones like the Roku TV[2] that has a 512MB-1GB RAM and an ARM Quad-Core processor. Depending on their application domain, they also use different types of communication protocols and technologies. For communication, they can use many different short-range technologies, such as IEEE 802.15.4, Bluetooth Low Energy (BLE), WirelessHART, Z-Wave, 6LoWPAN, RPL, CoAP, and MQTT (Message Queue Telemetry Transport) [12] or long-range ones like LoRaWAN, NB-IoT and Sigfox. IoT devices can also be found, due to their low costs, in domestic networks using the conventional TCP/IP stack over IEEE 802.11 (Wi-Fi).

IoT architectures can be divided into three layers: the collection layer, the transmission layer, and the processing, management and utilisation layer [12, 13]. In the collection layer, the main objective is to collect information about the physical environment. After the capture of information, the data is sent to the transmission layer, usually using gateways. The transmission layer is responsible to deliver the collected data to different external servers and applications. In this layer, the gateways use technologies such as Ethernet, WiFi, Hybrid Fiber Coaxial (HFC) and Digital Subscriber Line (DSL) combined with TCP/IP protocols to build a network that interconnects objects and transmit the collected data in a long range. Lastly, the processing, management and utilisation layer deals with analysing and processing information flows, sending the data to applications and servers, and providing feedback to control applications. Also, this layer is responsi-

[1] https://www.advanticsys.com/shop/mtmcm5000msp-p-14.html
[2] https://www.roku.com/en-gb/

ble for discovering and managing devices, filtering and aggregating data and information utilisation.

2.2 IoT Botnets

Due to their use in the composition of IoT services, technologies and standards, the IoT paradigm has the same security issues as mobile communication networks, cloud services and the Internet [12]. However, once they are simple and resource constrained, the traditional countermeasures and privacy enforcement cannot be used in IoT devices [14]. Thus, they can be compromised without much effort by a malicious user, and be used to perform several illegal activities, e.g., DDoS attack [15].

DDoS attacks consist of comprising a very large group of devices, usually scattered around the globe, which perform coordinated DoS attacks. This type of attack is characterised by performing precise attempts to compromise a service [2]. Generally, it achieves this goal by sending a large volume of packets that occupy a significant proportion of the available bandwidth and/or computational capacity of the target [16] [3].

Usually, DDoS attacks are controlled by a botnet, a network formed by infected devices remotely controlled. Botnets are composed of three main parts: the bots, which are the infected devices; the botmaster, which is the malicious user that controls the bots; and the C&C infrastructure, which is used to establish a communication channel between the botmaster and the bots [17, 18]. One point to note is that bots can be any device ranging from traditional desktop computers to mobile and IoT devices. By using a large collection of bots, the botmaster is able to carry out a diverse set of coordinated malicious actions such as DDoS attacks, generating spam and stealing sensitive information.

There are several types of botnets spread over the Internet. In this work, we focused on botnets designed for IoT devices that run Linux or BusyBox[3], i.e., programs found in some domestic IoT devices and exploited by botnets. Using the VirusShare[4] platform, we searched for malware samples for IoT architectures, such as Microprocessor without Interlocked Pipeline Stages (MIPS) and Advanced RISC Machine (ARM), and by names of IoT botnets found in [2]. We found five botnet families with working samples and the source code of the Mirai botnet[5]. The botnets used in this work are:

- **Mirai**: this botnet focuses on compromising IP cameras by performing a brute-force attack exploring the Telnet protocol. Its main malicious activity is to perform a range of DDoS attacks. Also, it uses a centralised C&C infrastructure via HTTP protocol. In its peak of activity, it had a network traffic of 1.1 Tbps [2];

[3] https://busybox.net/
[4] https://virusshare.com/
[5] https://github.com/jgamblin/Mirai-Source-Code

- **Hajime**: a botnet that focuses on the same IoT devices as Mirai and uses the same type of infection strategy. The purpose of Hajime is unknown, as it has only closed some vulnerabilities in IoT devices, i.e., it has not behaved maliciously so far. Its C&C infrastructure is fully decentralised by using the BitTorrent Distributed Hash Table (DHT) protocol, while also encrypting its messages with RC4. When a device is infected by this botnet, the terminal prints a message informing the infection to the user [19, 3];

- **Aidra**: this botnet compromises devices based on the following architectures: MIPS, MIPSEL, ARM, PPC, x86/86-64 and SuperH. Its infection process works in the same way as Mirai. This botnet malware opens a port on Linux-based devices or computers to wait for commands of the C&C server. The main malicious activity performed is DDoS attacks. A version of the botnet source code was released on the Internet[6] [19];

- **Bashlite**: the main targets of this botnet are Linux-based IoT devices. The device infection occurs through a brute-force attack on the Telnet port using default credentials. After the infection, the device performs DDoS attacks on C&C command. The C&C's IP addresses for communication are included in the source code, facilitating the task of monitoring the botnet communication. In its peak of activity, it had a network traffic of 400 Gbps [2];

- **Dofloo**: this botnet is found in regular OS for desktop computers, such as Windows and Linux, and IoT devices with MIPS and ARM architectures. It is used to launch several DDoS attacks following the instructions of the centralised C&C, using AES-encrypted messages. In its peak of activity, it had a network traffic of 215 Gbps. Also, it collects memory, CPU and network traffic data and send to the attacker [2];

- **Tsunami**: also targeting IoT devices, the infection occurs by downloading and executing infected files. The malicious activity of this botnet is to perform DDoS attacks. It uses an IRC channel and HTTP requests to communicate with the malicious servers controlled by the attackers. It changes the DNS server settings of the infected device [2];

- **Wroba**: having been found initially targeting Android devices [20, 21], Wroba migrated to IoT devices with ARM architecture. Its main purpose is to intercept or attack web banking activities, while infecting other devices. It connects with its C&C infrastructure using HTTP, but on Android devices it can also use SMS messages.

[6] https://github.com/eurialo/lightaidra

2.3 Intrusion Detection Systems

Intrusion Detection System is a program or mechanism that focuses on detection of attacks against a system or a network by analysing features in the network or in the host itself [6]. These systems can be divided into two major technology classes: network-based IDS (NIDS), which is dedicated to monitoring network traffic to detect attacks against various elements in a network, and host-based IDS (HIDS), which is positioned on an individual host and is dedicated to detecting attacks against only that host [7].

These systems use different types of methods to detect intrusions: signature-based, anomaly-based, and stateful-protocol analysis [22, 23]. Signature-based methods use a pattern or string that corresponds to information about an attack. These patterns and strings, i.e. signature, are compared with captured events to recognise a possible intrusion. This method is simple but requires up-to-date signatures to detect new attacks [24, 22].

Anomaly-based methods create profiles or models representing the normal or expected behaviour after capturing data about legitimate or regular activities, users or connections in a period of time [24]. These profiles and models are compared to captured events to recognise an intrusion or attack. The advantage of this type of method is the effectiveness to detect new and unforeseen threats, but there may be some alerts not related to intrusions or attacks, being consequence of a malfunction or a unusual behaviour. Lastly, stateful-protocol approaches know and trace the protocol steps, e.g., comparing requests with replies. Unlike anomaly-based approaches, they come with vendor-developed profiles for protocol steps, which are compared with observed events. This methodology is very effective with unexpected sequences of commands but is unable to inspect attacks using legitimate protocol behaviours [22].

The mentioned methods use different detection approaches to recognise an intrusion in an observed event: statistics-based, pattern-based, rule-based, state-based and heuristic-based [22]. Statistics-based approaches use statistics methods (e.g mean and standard deviation) to build a threshold to detect an intrusion. Pattern-based ones compare records and signatures of attacks to audit intrusions in captured events. Rule-based approaches use If-Then and If-Then-Else rules to build profiles of known intrusions. State-based approaches explore the use of finite state machines from network behaviours to identify attacks. Lastly, heuristic-based ones use methods based on biology, machine learning and artificial intelligence.

Recent trends in IDS can be found in works [23] and [25]. Supervised machine learning algorithms are widely used to build these solutions. Among them, the Random Forest algorithm is one of the most applied. The paper by Turrisi da Costa et al. [18] is an example. In this work, the Random Forest algorithm was used to analyse applications' syscalls to detect mobile botnets in Android devices. The Support Vector Machine algo-

rithm is another method frequently adopted. In [26], an IDS based on SVM was developed to detect denial of service attacks in mobile ad hoc networks (MANETs).

Deep Learning, a subarea of Machine Learning, has also been explored to build IDSs. Deep Learning consists of using artificial neural networks with a large number of layers (deep) that enable learning from experience, relying on a large amount of data [27]. An example of work that uses this method is [28], which developed a NIDS based on a deep learning algorithm referred to as autoencoders. Lastly, data stream learning is another class of Machine Learning algorithms being employed in intrusion detection. It consists of learning incrementally with new instances, without consuming significant memory or time. In [29], this method was applied to detect botnets by analysing NetFlow data.

2.4 Machine Learning

The study of ML is about how to develop algorithms that automatically improve with experience [30, 31]. This study involves different areas of knowledge, such as statistics, artificial intelligence, philosophy, information theory, biology, cognitive science, computational complexity and control theory [31].

Developing methods that can improve with experience requires tools to learn and also measure the learning, e.g., performance metrics, learning models, forms of mathematical problem representation machines can interpret and use. Learning methods are separated into two main types of approaches: supervised and unsupervised learning. Supervised learning methods require a specialist to know the correct answers in a specific problem, for example, a network manager who knows which network packet is made by a user or malicious program. This process is known as labelling, referring to the task of assigning labels to inputs indicating the expected output. From these labels, the ML algorithm is capable of associating different patterns with the answers given by the specialist [23]. Supervised learning techniques have two types of predictive functions: regression and classification [32]. In regression problems, the program output is based on continuous numeric values. In classification problems, the output is a category or class [31].

The second approach is the unsupervised learning, in which the expected output or even a expected behaviour are unknown. This approach consists of grouping similar behaviours that have not been discovered yet using the information presented in the data. For example, an unsupervised algorithm could be used to group network packets in a specific number of groups. Each group would contain similar packets based on their size, duration and protocol, which could allow the network administrator to find packets linked to specific events, e.g. DoS attacks.

2.5 One-class Classification

In traditional supervised classification ML techniques, data collected previously has to be associated to labels representing all the classes of interest to induce a model capable of detecting each class. This requirement for labelled data from each possible class can be problematic in some cases, e.g., when the labelling cost is too high or some class occurs much less frequently [33, 34]. When considering botnet detection, labelled data may be collected by a specifically designed testbed or honeypot. This requires manual work to set up vulnerable devices, and, accordingly some kind of access to botnets. Additionally, there is an inherited class unbalance problem when dealing with malware detection, where legitimate data is easily available and in far greater scale.

An emerging alternative is to use one-class classification techniques, which do not model all classes. Instead, data from a class of interest is used to create a model capable of yielding if a given new instance of data belongs to this class or not [33, 34]. In the context of botnet detection, after collecting legitimate behaviour data (which is more straightforward to obtain than malicious data), one-class classification algorithms induce a model that discerns if a given new instance is legitimate or not. Here, it is considered that when an instance does not belong to the induced model, it is associated with malicious behaviour. In this sense, one-class classifiers can be used as an alternative to traditional supervised classification algorithms to reduce significantly the labelling cost and the need for malicious data in the training phase. Likewise, they are easier to adapt, given that periodical model updates using new legitimate data demand less effort than updates with labelled legitimate and malicious data. In this work, we compare the performance of 4 one-class classification algorithms, namely Elliptic Envelope, Isolation Forest, Local Outlier Factor and One-class Support Vector Machine.

These four algorithms were selected by representing different approaches in one-class classification. The first algorithm, Elliptic envelope, was chosen due to its simplicity in detecting outliers. This algorithm uses a calculated ellipse that separates the outliers from the inliers. Next, representing ensemble methods, we picked Isolation Forest. Ensemble methods are used in many network security problems [25] being a possible solution for this work. We also wanted to have a clustering-based approach that could analyse the data using the distance between the normal instances, thus we chose the Local Outlier Factor. Lastly, we searched for an algorithm that has good results in detecting behaviour changes and we found the One-class Support Vector Machine, that was successful in detecting machine faults [35].

2.5.1 Elliptic Envelope

The first algorithm is the Elliptic Envelope [36], a simple method to detect outliers based on a known distribution. The algorithm models the data as a high dimensional

Gaussian distribution with possible covariances between the feature dimensions [37]. In other words, the algorithm tries to find an ellipse that contains most of the data. The data inside the ellipse is considered inlier and the outside points are outliers. This method uses the FAST-Minimum Covariance Determinate [36] to estimate the size and the shape of the ellipse. This FAST-Minimum Covariance Determinate uses the Mahalanobis distance, which is a measure of the distance between a point a and a distribution D, in conjunction with a covariance matrix to fit this ellipse. The parameter of this algorithm is the contamination, which indicates the fraction of points in the data the algorithm can discard to fit the shape. This is used since the presence of outliers in the data can cause some distortion on the fitted Gaussian.

2.5.2 Isolation Forest

The second one-class algorithm is the Isolation Forest (IF) [38]. This method detects anomalies instead of profiling normal data points. IF works building decision trees. The partitions are created by randomly selecting a feature and then selecting a random split value between the minimum and maximum value for this feature.

The main idea of this process is: outliers are less frequent and often bigger or smaller than the rest of the instances. The random partitioning will put these bigger or smaller elements, outliers, closer to the root of the tree, creating trees with shorter path length. Thus, if a element is in a shorter path of a tree, it is an outlier. In this method, an anomaly score represented by the following equation is used:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}, \tag{2.1}$$

where $h(x)$ is the path length of an instance x, $c(n)$ is the average path length of a unsuccessful search in the binary tree, n is the number of external nodes (a node with no children) and $E(h(x))$ is the average of $h(x)$ from a collection of isolation forest trees [38]. Every observation receives a score. If $s(x, n)$ is close to 1, this indicates an anomaly. A score smaller than 0.5 indicates a normal instance. Finally, if all instances' scores are close to 0.5, the entire sample does not have anomalies.

2.5.3 Local Outlier Factor

Another algorithm for one-class classification is the Local Outlier Factor (LOF). The algorithm estimates a score (named Outlier Factor), reflecting the level of abnormality of each observation contained in a dataset [39]. This algorithm works based on the idea of a local density. The k-Nearest Neighbours algorithm is applied in the data, and each data instance is given a locality, which is used to estimate the clusters' density. First, it is decided the number of neighbours, set in the variable k. Choosing an appropriate value for k is very important, once a small k has a more tighter focus, but has more errors when

dealing with noisy data. On the other hand, a large k can include all the instances that are outliers.

The algorithm uses a function called *k-distance*, which is the distance of a point to its *kth* neighbour. This *k-distance* is used for the calculation of the reachability distance (RD), which is the largest value out of the distance between two points and the *k-distance* of the second point, as showed on Equation 2.2. If a point a is in the k neighbours of b, $RD(a, b)$ is equal to the *k-distance* of b, otherwise it will be the real distance between a and b.

$$RD(a, b) = max\{k\_distance(b), distance(a, b)\} \tag{2.2}$$

This RD function is used on the calculation of another function, the local reachability density (LRD). In order to calculate the LRD of a point a, first it is calculated the RD of a to all its k neighbours ($N_k(a)$), followed by the average of the resulting values as presented on Equation 2.3.

$$LRD(a) = \left(\sum_{n}^{N_k(a)} \frac{RD(a, n)}{k} \right)^{-1} \tag{2.3}$$

Next, it is calculated the LRD of each point in the dataset. Each point compare its LRD to their k neighbours'. The LOF is the average ratio of the LRD of a point a to the LRDs of its closest points, k neighbours. As showed on Equation 2.4, if the LOF is greater than 1, the LRD of a point a is on average greater than the LRD of its k neighbours.

$$LOF(a, N_k(a)) \begin{cases} \approx 1 & \text{Similar density as neighbours} \\ > 1 & \text{Lower density than neighbours (Outlier)} \\ < 1 & \text{Higher density than neighbours (Inlier)} \end{cases} \tag{2.4}$$

2.5.4 One-class Support Vector Machine

Multiple one-class classification techniques have been developed throughout the years, with the One-class Support Vector Machine (OSVM) [40] being successfully used in multiple domains, such as, document [41] and handwritten signature [42] classification, and machine fault detection [35].

The OSVM is an extension of the SVM created by Vapnik [43], which is a traditional supervised ML algorithm used for classification problems. The algorithm works by creating a hyperplane (n-dimensional plane) that better separates two different classes. The SVM tries to optimise the separability of the data, searching for instances that are within the borders that split the data. Those instances are called Support Vectors. For the

use of the SVM, we need to process the data in kernel functions. The kernel functions are responsible for taking non-linearly separable data and map them to a higher dimensional plane that can be linearly separated.

Unlike the traditional SVM, the OSVM uses the hyperplanes to create boundaries around a region that better contains all training data. By doing so, the OSVM is capable of identifying if an instance is inside the area or not. To create these hyperplanes, the OSVM optimises the following objective function:

$$\min_{w,\xi,\rho} \frac{1}{2} w^T w + \frac{1}{\nu N} \sum_{i=1}^{N} \xi i - \rho \tag{2.5}$$

subjected to $w^T \phi(x_i) \geq \rho - \xi_i$ and $\xi_i \geq 0$, where $x_i \in \mathbb{R}^p$ is an instance of the training set, w is the weight vector of the hyperplane in the inner product space, $\phi(.)$ is a mapping from the original p-dimensional feature space to an inner product space, ξ_i's are penalty terms for error, ρ is a bias term, and $\nu \in (0, 1]$ is a parameter that poses an upper bound on the fraction of outliers in the training set.

The decision hyperplane can be represented by $g(x) \equiv w^T \phi(x) - \rho = 0$. The solution of the convex optimisation problem of the objective function finds the decision hyperplane, and the corresponding decision function $f(x)$:

$$f(x) \begin{cases} w^T \phi(x) - \rho \geq 0 & \text{if } x \text{ belongs to the set} \\ w^T \phi(x) - \rho < 0 & \text{if } x \text{ is an outlier} \end{cases} \tag{2.6}$$

In this work, it is used the radial basis function (RBF), the polynomial and the linear kernel to map the input space to a higher dimensional space. The RBF kernel is described by the following equation:

$$\phi(x) = \exp(-\gamma ||x - x'||^2), \text{where } \gamma > 0. \tag{2.7}$$

The parameter γ is the influence in the training samples. Next, the polynomial kernel is described by the equation:

$$\phi(x) = (\gamma \langle x, x' \rangle)^d, \tag{2.8}$$

where d is the degree function passed as a parameter. Lastly, the linear kernel is represented by the equation:

$$\phi(x) = \langle x, x' \rangle. \tag{2.9}$$

2.6 Related Work

In this section, we summarise the previous work related to the main contributions of this work: the intrusion detection system for IoT and the dataset containing data on IoT botnet attacks.

2.6.1 Prior Work on Intrusion Detection for IoT

Most of the works on intrusion detection for IoT are based on simulated networks using the protocol 6LoWPAN [12]. The two most common operating systems deployed at these 6LoWPAN-based devices are Contiki and TinyOS. Although these works have promising results, exploring host-based, network-based and hybrid IDS, their focus on 6LoWPAN networks may prevent their application in networks based on other IoT protocols, such as Bluetooth Low Energy, and the conventional TCP/IP stack.

One of the most mature IDS for 6LoWPAN networks was dubbed SVELTE [6]. It is composed of several modules to protect devices from different attacks. The first module of the system uses the routing protocol RPL (Routing Protocol for Low power and Lossy Networks) to map the network topology and discover the nodes within the reach of IDS agents that are distributed over the network. Then, the distributed IDS agents check the packets of their neighbours. This process aims to protect the devices from insider attacks. Another module is a distributed firewall deployed on the devices, which protects them from an outsider attack, closing ports and blocking addresses. SVELTE was tested in a network of Contiki-based devices.

In [7], an IDS is developed for TinyOS. The proposed IDS has some watchdogs devices to analyse the packets being sent by their neighbours. This allows the system to detect traffic-based attacks and abnormal behaviour on the network. The IDS also enables the creation of rules and countermeasures that must be executed by the devices upon the detection of an intrusion. A limitation of their approach is that the rules may not have the needed complexity for responding accurately to a network attack, and the communication between the devices and the management module is insecure.

Other works in IDS for IoT focus on conventional TCP/IP stack and IoT domestic devices. Most of these works are NIDS intended to be a module in a router or gateway, using whitelists, machine learning methods or pre-defined rules. In [11], it is proposed an IDS named Heimdall. This IDS uses a whitelist to prevent IoT devices from connecting to malicious address, avoiding communications with botnets C&C or private data leaks. Heimdall is intended to be a module in the local router, acting as a gateway for IoT devices, using third-party analysis of malicious addresses from organizations such as VirusTotal, Metadefender, and VirScan, combined with an auditor and DNS validation. They tested this approach on real domestic IoT devices and the method is effective against the attacks

developed by the researchers, as well as has minimal overhead. However, tests with real botnet attacks were not carried out, and the IDS depends on the maintenance of the third-party systems.

Other works such as [44] and [45] aim to build anomaly-based systems to detect IoT botnets. These works present techniques that model the legitimate behaviour of IoT devices. This type of approach can be effective due to the systematic behaviour of IoT devices, which usually perform specialised tasks. Thus, behaviour deviations are detected as malicious, regardless of the botnet.

In one of these works, Meidan et al. [44] proposed the N-BaIoT, which is a NIDS that uses deep autoencoders to detect botnets in IoT devices. They use the network traffic of IoT devices to build a model of legitimate behaviour and detect any anomaly. They tested the approach on two botnets, Mirai and BashLite, and without using labelled data, they achieved great results in detecting the attacks, with low false positives rate. Despite the great results, the use of deep autoencoders can be computationally costly even to a gateway and demands large amounts of data to train the model. The work proposed by [45] explores the construction of an IDS to protect Linux routers, a very popular target in recent years. They tested three different types of anomaly detection techniques, Principal Component Analysis (PCA), One-class SVM and a naive detector using n-grams, to analyse syscall data from routers. They used two botnets to test their approach, MrBlack and Mirai, in simulated routers, and all tested methods presented good results. However, it can be difficult to find the best features among syscalls during data pre-processing, due to the large number of syscalls generated by multiple programs running in the system.

Overall, different issues were found in these works such as the lack of testing in real devices or with different botnet samples, dependency on third-party systems, use of computationally costly methods, and need for large amounts of data to train the models. Also, the analysed works did not propose many host-based methods and did not explore resource consumption data to detect botnets. Considering these gaps, we proposed an approach for botnet detection in IoT devices that does not need malicious labelled data to build a detection model and relies on host-based data such as CPU and memory utilisation. Additionally, the approach does not affect negatively the device operation and is designed to be used in more robust IoT systems, such as the Raspberry Pi, Odroid, Roku TV and Google's Chromecast.

2.6.2 Prior Work on Datasets

There are several exemples of datasets containing network traffic, e.g., CAIDA[7] (Center for Applied Internet Data Analysis), KDD [46] (Knowledge Discovery and Data

[7] http://www.caida.org/home/

Mining), DARPA[8] (Defence Advanced Research Projects Agency), the N-BaIoT Dataset [44] and CIC (Canadian Institute for Cybersecurity) [47], which have been used by researchers for many purposes, such as, to test the effectiveness of different IDS and provide benchmark resources for the comparison of proposed solutions.

The CAIDA repository consists of datasets with different types of network data collected from a diverse range of situations. Some packet fields are anonymised or completely removed for security reasons. Also, some packets are not labelled. Nonetheless, these processes can have a negative impact on the evaluation of an IDS [47, 48].

Another public repository is the DARPA dataset, which was built by the Lincoln Laboratory at the Massachusetts Institute of Technology (MIT) with the objective of evaluating IDS. The datasets in this repository are in the *tcpdump* format and include all the packets payloads, which were collected from real and emulated machines, and where attacks were only performed on real machines. However, the experimental environment used to build the dataset was not made public and the attack methods are considered outdated [49].

The KDD repository has a network dataset as well, which is mainly used for benchmark tests of IDS. It contains data collected from six machines, where three were real machines and the others were simulated. Each simulated machine has a different OS. The three real machines were used to generate background traffic. In addition, a sniffer captured and stored the traffic in the *tcpdump* format. As DARPA datasets, the KDD dataset is quite old, so the attacks and the background traffic do not match to current networks reality.

The N-BaIoT [44] was generated from network data collected from 9 domestic IoT devices. They were connected to different access points, which were wire connected to a central switch. To collect the network packets, the authors performed a port mirroring on the central switch and recorded the data using Wireshark. The attacks present in the dataset were based on two botnets, Bashlite and Mirai. Although the experimental setup has some positive aspects such as including different attacks and a wide range of real domestic IoT devices, the dataset only brings some selected traffic statistics, not sharing the captured PCAP files. This may limit its adoption by other researchers.

CIC provides a dataset and scripts that can be used to generate legitimate and malicious traffic on-demand [47]. Their dataset is composed of network traffic collected from several machines, which performed regular activities and were also infected by different malicious applications. Although it meets the criteria of a good dataset described in [47], it may not represent a real botnet behaviour [48], as they used a botnet malware developed by them. In this sense, to have a realistic scenario, we use only samples of real

[8] https://www.ll.mit.edu/ideval/data/

and active botnet malware for IoT devices.

Overall, we can observe that CAIDA, KDD and DARPA repositories focus on benchmark tests, but they use obsolete attack types and tools. In addition, they do not have all the necessary network traffic information, e.g., destination IP addresses, payloads, and labels. The N-BaIoT dataset was generated from an experimental setup with a wide range of real IoT devices, but the shared data includes only some selected statistics about the traffic. The CIC dataset is newer than the other ones, hence it includes more recent attacks such as botnets and HTTP Denial of Service (DoS). However, as CAIDA, KDD, and DARPA datasets, the CIC dataset does not provide data related to IoT. Considering these previous works, we built an IoT dataset that fills the pointed gaps, such as using real samples of botnet malware, including legitimate traffic.

3 IOTDS: A HOST-BASED BOTNET DETECTION SYSTEM FOR IOT

In this chapter, the IoTDS is described. First, it is presented a system overview, Next, the architecture is presented, detailing its two main components, the Agent and the Management Console. Lastly, the two phases of the detection process, the Model Induction and the Continuous Analysis, are explained.

3.1 IoTDS Overview

Since IoT devices are very specialised, running simple, repetitive and well-defined tasks, these devices behaviour are supposed to present a clear level of regularity as long as they are not compromised. Likewise, a malicious software that compromises a IoT device should alter the device's behaviour. Therefore, it is possible to use an one-class classifier to model the device's behaviour and detect anomalous events, avoiding the effort demanded by the collection and labelling of malicious data to train the classifier.

IoTDS is a host-based system. Running the system in the IoT device, instead of at some point of the network, allows the collection of data about the device's resource consumption, which can be more sensitive to the presence of bot malware. Also, by relying only on resource consumption data, the IoTDS ability of detecting malicious activity is not hindered by the encryption of malicious traffic. Lastly, IoTDS can be pre-installed in IoT devices, given protection to these devices out of the box.

Broadly speaking, IoTDS works creating models of the legitimate behaviour of the IoT devices and detecting any anomalies using host-based data. If the behaviour of the device presents an anomaly, the device sends an alert to a central server. After a given number of consecutive alerts, the system tries to stop the unusual behaviour executing a reboot of the IoT device's system.

IoTDS is divided into two phases, Model Induction and Continuous Analysis. These two phases are executed for each IoT device. Also, the system architecture is divided into two elements: Agent, installed in the IoT device and responsible for analysing the device's data, and Management Console, installed in a separated computer server, where the alerts are stored and an overview of the devices situation is provided to the network administrator.

Two main challenges coming from our design choices were addressed in this work. Being a host-based solution, the system cannot consume a large amount of the device's resources, since that would be harmful to other functionalities. Bearing that in mind, we move the heaviness of inducing a behaviour model to a centralised server with more

computational power, which hosts the IoTDS Management Console. Hence, the IoT device runs an already trained one-class classifier, which may be done at a minimum cost of computational power.

Processing data in a separated computer brings challenges regarding channel security to the proposed system. Since the IoTDS Agent captures host-data about the device, it is necessary to build a secure communication between the Agents and the Management Console. In the proposed approach, the HTTPS protocol is used to protect the information travelling between the Agents and the Management Console. HTTPS provides confidentiality and integrity, as well as allows the authentication of the end points [50].

The IoTDS is designed to offer the following advantages:

1. *Distributed architecture*: by being a distributed system, the IoTDS does not need a dedicated and centralised computer to analyse data constantly coming from a large number of IoT devices. This saves network resources and improves system scalability;

2. *No need for Internet connection*: the system does not need any Internet connection or use third-party online software to detect IoT botnets, such as whitelists and other software to verify valid IP addresses/ports;

3. *No use of malicious data during the training step*: the system does not need the collection or use of malicious data to induce the classification model during the training step. It only has to capture legitimate data on the IoT device's behaviour to do it;

4. *No use of traffic analysis*: the system does not analyse traffic data. When botnet traffic is encrypted end to end, network-based solutions can face difficulties to analyse the traffic.

3.2 IoTDS Architecture

The approach is divided into two elements: the IoTDS Agent and IoTDS Management Console. As presented in Figure 1, the Agent is installed in the IoT devices and the Management Consoled in a separated computer connected to these devices. Next, more details about these two elements are provided.

3.2.1 IoTDS Agent

The IoTDS Agent is installed in the IoT devices and is responsible for capturing and analysing the device's data, sending alerts to the Management Console. This element

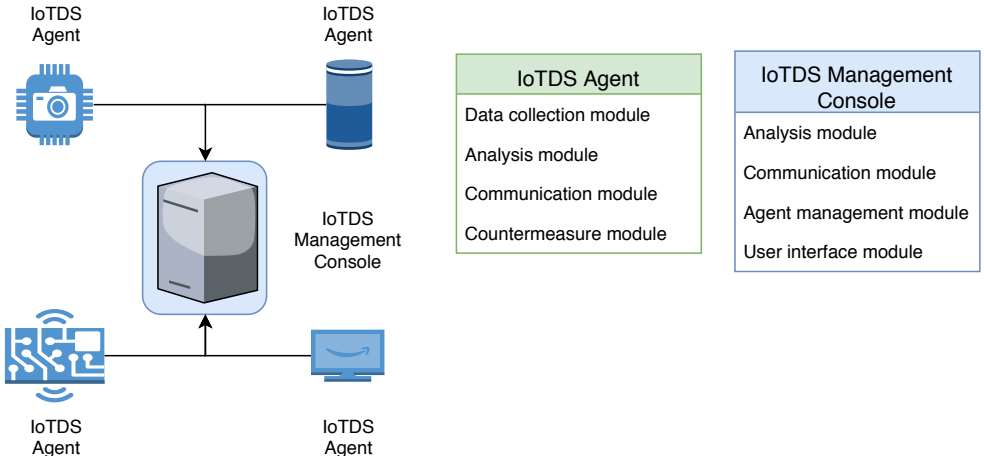

Figure 1 – IoTDS elements.

has the following modules: the Data Collection module, Analysis module, Communication module and Countermeasure module. The Data Collection module is responsible for collecting the following data about the system behaviour, which are CPU and memory usage, number of running tasks and CPU temperature. This module is used in both the Model Induction and Continuous Analysis phase.

The data collected by this module is processed to be used by the Analysis module or sent by the Communication module to the IoTDS Management Console. How data is processed depends on the purpose of its collection. During the Model Induction Phase, the data is collected each second during an interval of y minutes, stored, and then sent to the Management Console, which will build the behaviour model. In the Continuous Analysis phase, this data is consumed by the Analysis module to be classified.

The Analysis module uses a behaviour model sent by the IoTDS Management Console to analyse data collected in real time and decide whether an anomaly is occurring on or not. The Communication module is responsible for the communication between the Agent and the Management Console, sending collected data and alerts, as well as establishing a secure channel for this communication, checking the CA (Certificate Authority) of the Management Console. Finally, the Countermeasure module is responsible for rebooting the system when it receives the system's reboot command.

3.2.2 IoTDS Management Console

The IoTDS Management Console is responsible for creating behaviour models for the IoTDS Agents, storing alerts and information, and presenting alerts and information about the IoT devices to the network administrator. The Management Console is installed in a separated server connected with the IoTDS Agents. It can be installed in the gateway of an IoT network, which has access to all devices and enough computational power to

run all the Management Console's modules. Also, the IoTDS Management Console can be installed in a remote server connected through the Internet, or even in a local desktop computer connected to the private network. The modules that compose the Management Console are the Analysis module, the Communication module, the Agent Management module, and the User Interface module. The Analysis module is responsible for creating behaviour models for the IoT devices.

Since IoT devices have constrained computation power, they cannot build a behaviour model without compromising their functionality and battery power because this is a heavy task. Also, in a centralised server, it is quicker to build a behaviour model and it is easier to change the behaviour model that is in use. The Agent Management module has the functionality of registering the active IoTDS Agents, their alerts and their information. This module works when an Agent sends a request for a behaviour model. Then, the Agent Management module stores the applicant IP address and the device's name. Also, this module is called when an Agent sends an alert. The Communication module has the same responsibilities as its homonym in the IoTDS Agent. The IoTDS Management Console has CA certificates to create a HTTPS based channel, which offers integrity, privacy and trust. The adoption of HTTPS also prevents replay attacks, meaning that an attacker cannot, for example, take a reboot command and send it again to the IoTDS Agent. Finally, the User Interface module is responsible for presenting information about the generated alerts and active agents to the network administrator.

3.3 Process Details

This section presents the details on how the IoTDS carries out the botnets detection. This process is divided into two phases, the Model Induction and the Continuous Analysis.

3.3.1 Model Induction Phase

The Model Induction phase, illustrated in Figure 2, consists of gathering host-data and building a classifier. First, the IoTDS Agent collects the memory and CPU utilisation, the number of running tasks and the CPU temperature at every second for an interval of y seconds. Then, these observations are processed according to a time-window to extract the data instances to be used to induce the behaviour model. Considering a time window of s seconds, the value $x_s[j]$ to a feature j at the time window s will be the arithmetic mean of the s observations collected for j during the s seconds. All the extracted data instances for the y-second interval are included in a CSV file, which is sent to the IoTDS Management Console in a HTTP-POST request through a HTTPS-based connection.

Then, in the Management Console, all features for each instance are scaled accord-

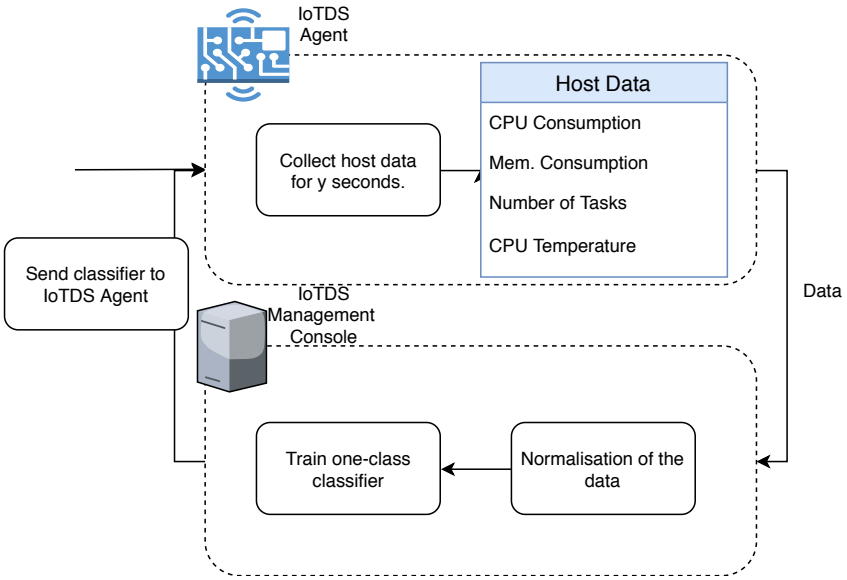

Figure 2 – Description of the Model Induction phase.

ing to their minimum and maximum values between a range from 0 to 1, defined by the following equation:

$$x_s[j] = \frac{x_s[j] - min(x[j])}{max(x[j]) - min(x[j])}, \forall s \in y, \forall j \in J \tag{3.1}$$

where s corresponds to a given instance in a second, y to all instances collected during that period, j to a feature in the J feature space, and $x_s[j]$ to the value present in feature j for the instance s.

Once the instances are normalised, the Management Console trains the classifier, which is stored in a file and sent in response to the IoTDS Agent's previous request for the model.

3.3.2 Continuous Analysis Phase

The Continuous Analysis phase, presented in Figure 3, consists of using the induced model to monitor the device and try to prevent IoT botnets from running in the system. The IoTDS Agent first checks whether there is a classifier in the system. If the Agent does not have a classifier induced by the Management Console, it starts the Induction Model phase. After this check, the system starts a trust factor counter n with zero. Next, the Agent collects the observations to produce a new data instance according to the s second time window used to induce the model. This new instance undergoes the same normalisation step described in Equation 3.1, except that this time, the $max(x[j])$ and $min(x[j])$ are the same as the ones used when the classifier was induced. After that, the approach decides whether this new instance is a legitimate behaviour or an anomaly. If it

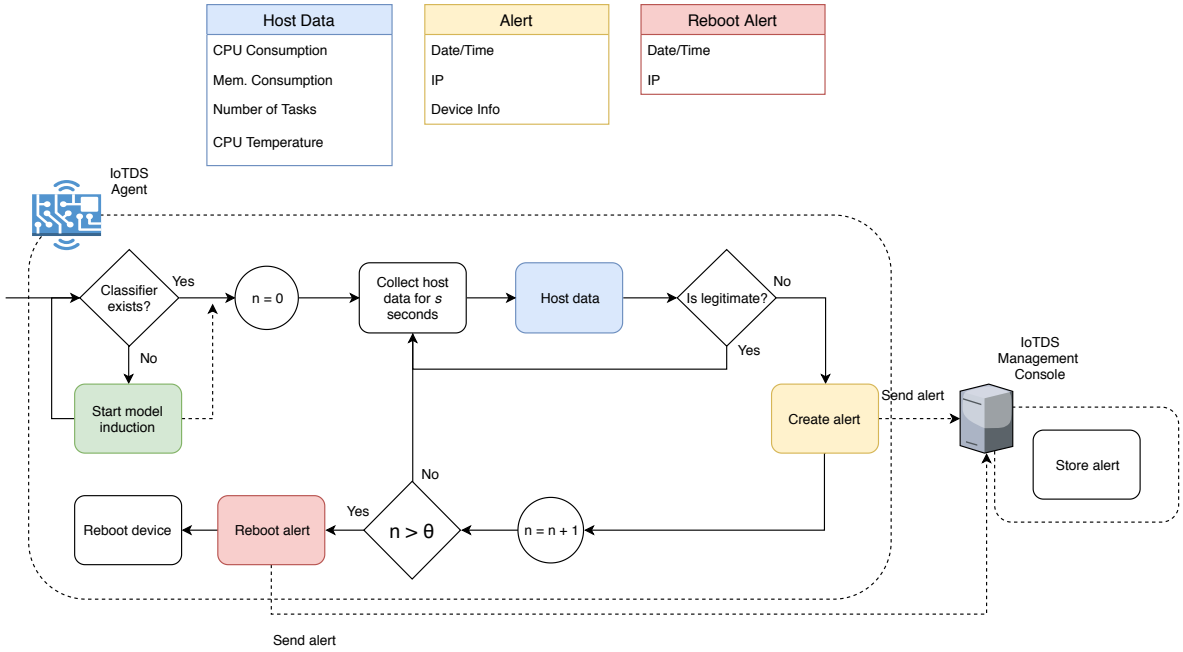

Figure 3 – Description of the Continuous Analysis phase.

is the latter, the device raises an alert.

The alert is sent to the IoTDS Management Console with information about the device's IP address, information about the resources utilisation and the timestamp of the alert. Once the IoTDS Management Console receives the alert, it is stored and presented in the user interface for the network administrator analysis. After the alert is sent, the IoTDS Agent increments the trust factor n by one, and resumes the host data analysis. The objective of the trust factor is to avoid generating an excessive number of alerts during an infection. If a device is compromised, the resource consumption data has its behaviour changed for as long as the malware operates. The Agent detects this unusual behaviour every time it analyses the devices' data, generating an alert at every new iteration. The trust factor takes place to control this situation. When the trust factor reaches a given threshold θ, the Agent tries to interrupt the malware operation. In this work we defined θ equal to three.

Most of IoT botnets, such as Mirai, BashLite and Hajime run in the device's RAM and, as a consequence, are not persistent. Thus, a system reboot is effective in removing the malware. When the trust factor reaches θ, the IoTDS Agent sends a reboot warning to the IoTDS Management Console informing the device's IP address and the timestamp, and then it reboots the system. When the device restarts, the Agent resumes the Continuous Analysis phase.

4 DATASET

This chapter presents the environment developed to accomplish one of the goals of this work, which is to build a dataset with host-based data collected from an emulated IoT device infected by botnet malware. The goal also is to provide a dataset that can be useful for researchers that are interested in IoT protection tools such as IDS but do not find a public IoT dataset to develop their work.

To build the experimental environment and, consequently, the dataset, works that proposed IDS for IoT were reviewed, analysing the IoT devices they used in their tests. As a result, it was decided to use a Raspberry Pi in our network. Next, profiles of operation for the Raspberry Pi were defined, emulating real devices such as multimedia centres and surveillance cameras. Lastly, it was created a network scenario to support the dataset generation.

4.1 Device Selection

Works in the literature that address intrusion detection for IoT do not focus on a unique device or type of device to test their approaches on. Thus, there is no a type of device that is considered a standard for IoT environments. This fact can be explained by the non-standard definition of the IoT itself. The IoT has a long list of definitions, which usually share only a few characteristics such as the use of devices with low-cost, limited resources and wireless communication capabilities. Table 1 presents some examples of devices used in previous work that addressed intrusion detection for IoT, since our focus is to contribute to this research field.

Table 1 – IoT devices found in previous work.

Reference	Device(s) Used
[7]	Cooja Motes
[11]	Amazon Dash Button, Arlo Home Security System, August Smart Lock, Lifx Smart Bulb, Nest Thermostat, Raspberry Pi
[51]	Ambient Weather, Network Camera
[52]	Philips Hue
[53]	Raspberry Pi
[54]	

As presented in Table 1, the devices range from very simple simulated boards (e.g.,

Cooja Motes) to everyday objects transformed into IoT systems (e.g., Phillips Hue lamps and Amazon dash buttons), which are made by different companies and use different types of software to control them. Among the devices used in previous work, the Raspberry Pi was chosen because it is a customisable general purpose device with good computational power, but still limited to the IoT constraints. This device, which was used in [11, 53, 54], has Wi-Fi and Bluetooth connectivity, is cheap, and supports a wide variety of OS, e.g., Linux and Windows. As commented by [11, 55], the Raspberry Pi is a device that can be applied in several fields and regarded as a small, powerful and compact device. Also, it supports additional hardware, e.g., camera, and is energy efficient. The specifications of the Raspberry Pi used in this work will be presented in Section 4.3.

4.2 Device Profiles

Raspberry Pi is a general purpose device and can be used to develop multiple IoT devices. Therefore, considering the characteristics of the IoT devices that could be targeted by the IoT botnets listed on Section 2.2, three IoT device profiles were defined and implemented in the Raspberry Pi. The profiles created are:

- **Multimedia Centre (MC)**: found in today's living rooms, a multimedia centre is a device that consumes streams of video such as movies or TV shows, to transform regular TVs in smart TVs. In addition to the use of video content, it also accesses an application store for updates and installation of other programs. The task of rendering video combined with the heavy traffic generated by video streams make this device to use a significant portion of its resources during its operation;

- **Surveillance Camera with Additional Traffic (ST)**: used both on the inside and outside parts of houses and companies, the operation of surveillance cameras is mostly characterised by the transmission of a video stream to a computer or station. However, this kind of device can also present traffic from other protocols. Telnet or SSH connections can be established to check if the device is working, its temperature, and the running processes, for example. These cameras may also include configuration web pages, which users access from their browsers to set up the parameters of cameras operation. As the device do not have to render video before transmitting, it consumes fewer resources than the multimedia centre;

- **Surveillance Camera (SC)**: this profile is similar to the ST profile, but it does not include interactions with users through Telnet, SSH, and configuration web pages. Having the video transmission as their single task, devices of this profile consume fewer resources than those of ST profile.

During all the experiments with the proposed profiles, it was executed the *tcpdump*[1] software to capture network packets, the *top* task manager program to capture CPU and memory usage and the number of processes running, and the function *vcgencmd* of the Raspbian OS to capture the CPU temperature. The video streams of ST and SC profiles were made using a VLC player[2]. In the MC profile, VLC was also used to consume video streams, and a Chromium Browser was employed to consume video and access the Chrome Web Store.

All the three profiles have Telnet and SSH ports open and a web server installed. This web server hosts the Damn Vulnerable Web Application (DVWA) software[3], which, in these devices, emulates a vulnerable web page used as an interface for device configuration. However, only devices of ST profile have legitimate traffic related to these three services. These three profiles were created to emulate an environment with devices that have different levels of resource consumption. SC devices show the lowest levels under normal conditions, while MC devices show the highest ones. Resource consumption in ST devices is higher than in SC devices and lower than in MC devices. The different profiles also allow the generation of more diverse traffic. Consequently, it provides more possibilities of analysis, such as the possibility of observing the impact of botnets in devices with different levels of resource consumption and multiple types of legitimate traffic.

4.3 Experimental Environment

After defining the device profiles that were going to be analysed, a network environment was created to support them. This environment provided Internet access and wireless communication for the IoT device. The network components present in the environment consist of a switch with Ethernet and Wi-Fi interfaces, four computers (*Machine 1*, *Machine 2*, *Machine 3*, and *Gateway*) connected to the switch through Ethernet, and one Raspberry Pi model 3B connected to the switch via Wi-Fi. The *Machine 2* provided two virtual machines, with the first one hosting a Web Server and the second being a client, referred to as *Device Admin*. Detailed specification of the environment is presented in Table 2.

Figure 4 details the network environment used to generate the dataset. The *Gateway* was used to provide Internet access and DHCP functionality to the network. The *Machine 1* hosted a DNS server that was used by the Mirai botnet. The same machine also hosted a VLC client that consumed the video stream generated by the SC/ST profiles and a VLC server that generated a video stream for the MC profile. The *Machine 2* hosted the *Web Server* and the *Device Admin*. The *Web Server* was responsible for

[1] Http://www.tcpdump.org/
[2] https://www.videolan.org/
[3] http://www.dvwa.co.uk/

Table 2 – Experimental environment components description.

Component	Specs	Operating System	Virtual/Real
Raspberry Pi	Ram: 1 GB CPU: ARM 1.2 GHz	Rasbian Stretch (Debian)	Real
Machine 1	Ram: 4 GB CPU: Intel i5 3.2 Ghz	Ubuntu 17.04	
Gateway			
Machine 3	Ram: 8 GB CPU: Intel Xeon 3.1 Ghz		
Device Admin	Ram: 1 GB CPU: 1 Ghz		Virtual
Web Server			

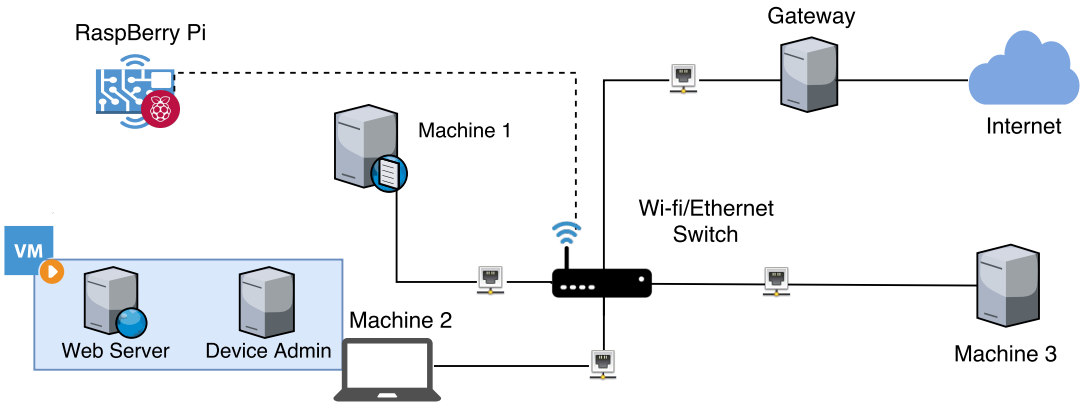

Figure 4 – Experimental environment network topology.

providing a static web page, which was accessed by the *Raspberry Pi* in all the profiles to emulate the consumption of web services. The *Device Admin* emulated the transactions of an administration software used to control and set up the emulated camera in the ST profile. To do so, it interacted with the *Raspberry Pi* through Telnet, SSH, and the Web configuration page. The *Raspberry Pi* was responsible for running the profiles presented in Section 4.2. Lastly, the *Machine 3* was responsible for emulating an attacker on the network, which infected IoT devices with botnet samples. Particularly for Mirai, this machine also hosted a C&C server, which could be used to make the *Raspberry Pi* launch DoS attacks against selected targets.

4.4 Dataset Construction

In this section, it is presented the experiment performed to create the dataset, using the environment described in Section 4.3. The goal was to provide two types of logs: one related exclusively to legitimate activities, and the other one containing data about both legitimate and malicious activities.

First, each profile defined in Section 4.2 was executed for one hour without any infection. The objective was to collect data exclusively related to legitimate activities. In

the ST and SC profiles, the *Raspberry Pi* transmitted a video stream to the *Machine 1* using VLC over HTTP through the port 8080. The video resolution was 80x40 pixels, and the transmission lasted for 20 minutes, with a total size of 4 MB. In the MC profile, the *Raspberry Pi* consumed videos from YouTube and Twitch[4], and a high definition video from the *Machine 1*. It also accessed the Chrome Web Store, emulating searches for new applications. In all the profiles, the *Raspberry Pi* accessed a web page hosted by the *Web Server* at 1-minute intervals, emulating situations in which the IoT device consumes web services. Particularly in the ST profile, the *Device Admin* interacted with the *Raspberry Pi* at time intervals of nearly 5 minutes through Telnet, SSH, and the web configuration page. During the execution of each profile, the following data was collected in the *Raspberry Pi*: CPU and memory consumption, CPU temperature, and the number of tasks running. All the network packets transmitted and received by the *Raspberry Pi* were also captured.

After the acquisition of data exclusively related to legitimate activity, data was captured on situations that combined legitimate and malicious behaviour. To generate this part of the dataset, the infections were organised into three categories: *Type 1*, *Type 2*, and *Type 3*. The number of samples and botnet families running on the device was varied to have more diversified data. Table 3 presents the botnet families and the number of samples running in each profile and method of infection.

In the *Type 1* infection, three botnets were selected, and they were installed one at a time in the *Raspberry Pi*. The objective was to observe their behaviour without the interference of any other malware. The first botnet selected was Hajime, because its main purpose is unknown and then it would be better to analyse its behaviour without the presence of other botnets. Botnets Aidra and Bashlite were selected because they have the highest number of samples. Thus, combining them with other malware might degrade the *Raspberry Pi* performance, hindering the tests. To infect the *Raspberry Pi* with these malware, the *Machine 3* connected to the device via SSH using Rasbian default credentials (Raspberry:Pi). Then, malware samples were transferred to the device through the established connection.

The *Type 2* infection was focused on Mirai botnet. The Mirai source code is available on the Internet, making it possible to create a complete scenario with the malware and the C&C server in our environment. Having access to a C&C server instance, it was possible to control all the life cycle of the botnet, from the infection to the DoS attacks against the victims. To infect the *Raspberry Pi* with the Mirai malware, the device from the *Machine 3* was accessed via SSH using the default credentials. Next, the Mirai C&C server was deployed at *Machine 3*. After having infected the device, the C&C command-line interface was used to make the *Raspberry Pi* launch DoS attacks against the *Gateway*,

[4] http://twitch.tv/

the *Machine 1*, and the *Web Server* at *Machine 2*. The duration of the attacks was short to simulate the malicious user testing the botnet's functions.

In the *Type 3* infection, the objective was to observe the behaviour of the device when it is infected by multiple botnets. Therefore, in this infection type, all botnets that were not used in previous ones were selected. Also the Mirai botnet was included to check whether the most popular IoT botnet would dominate or be dominated by its competitors. The infection process was carried out as in previous infections, i.e., via SSH connection.

Each type of infection was executed in each profile for one hour. After the end of each execution, all data and the OS were erased from the *Raspberry Pi* to avoid any contamination among the multiple types of infection.

Table 3 – List of infections made in the Raspberry Pi profiles.

Infection Type	Profile	Botnet(s)	Method of Infection	Number of Samples
1	MC	Hajime	SSH	4
	SC	Aidra		11
	ST	BashLite		
2	MC	Mirai	SSH and Telnet	1
	SC			
	ST			
3	MC	Mirai, Doflo, Tsunami,Wroba		28
	SC			
	ST			

Lastly, all the network traffic collected at the *Raspberry Pi* was labelled. Using the *Nfdump* software, NetFlow files were created from all the captured network packets. After that, all flows composed of packets that were captured during the one hour of execution without infection were labelled as legitimate.

To label the malicious traffic, firstly, all the flows with packets that were collected when the *Raspberry Pi* was infected were separated. Then, flows with IP addresses that did not match addresses of legitimate components of our network or known services such as YouTube, Twitch and the Chrome Web Store were labelled as malicious. The remaining flows were labelled as legitimate.

4.5 Dataset Files Description

Table 4 presents the details about all files in the dataset. File names with suffix "L" refer to the data collected when the *Raspberry Pi* was not infected. On the other hand, those marked with "I" contain data collected during infected periods, including legitimate and malicious activities. For each file name presented in Table 4, there are actually three files. The first one is a CSV file with the host data collected (CPU and

memory consumption, CPU temperature, and the number of tasks). The second file is a CSV file that contains IP flow data labelled as malicious or legitimate. The third file is a PCAP file with network packets captured at the *Raspberry Pi*.

Files regarding the testbed are provided as well. They consist of the image (ISO file) of the *Raspberry Pi* system used in the experiment, with the scripts to capture its host data and tools to create network traffic, the malware samples used in this testbed, and other network modules.

Table 4 – Files that compose the dataset.

File Name	Description	Profile	Infected by	Malicious activity?
MC_L	One hour of legitimate Netflow/Host Info from profile	MC	No infection	No
SC_L		SC		
ST_L		ST		
MC_I1	One hour of legitimate and malicious Netflow/Host Data from profile	MC	Type 1	Yes
SC_I1		SC		
ST_I1		ST		
MC_I2		MC	Type 2	
SC_I2		SC		
ST_I2		ST		
MC_I3		MC	Type 3	
SC_I3		SC		
ST_I3		ST		

5 RESULTS AND DISCUSSION

In this chapter, we present the results of the proposed approach. First, we present the technologies used to implement the system. Then, we divide the evaluation into two parts. The first one is related to the predictive performance of our approach. In this part, we execute the one-class algorithms discussed in Section 2.5. Our objective is to check which algorithms have the best performances and whether at least one of them can show a high detection rate along with a low false positive rate. In the second part, the impact of the system in the IoT device is studied. CPU and memory utilisation, CPU temperature, and energy consumed are observed to analyse how much impact the system has on the host main resources.

5.1 System Implementation

In this section, we describe how the IoTDS was implemented for the tests. The entire system was implemented in Python 3. The one-class classifiers were implemented using the Python SkLearn library 0.20 [56]. The Agent uses the Linux *top* program, which presents the utilisation of CPU/memory and the number of running tasks. This program is available in OS of some devices and in *busybox*, which is present in surveillance cameras. The Raspberry Pi's program *vcgen* was also used to collect information about the CPU temperature.

The communication between the Agent and the Management Console was made by using the library Python Requests[1], which allows the communication between HTTPS endpoints. The IoTDS Management Console was built using the framework Flask[2]. The system also uses CA certificates for HTTPS communication. The Python Joblib[3] library was used to export the induced classifiers to the Agents. All the alerts are sent as HTTP forms to the Management Console, and the reboot command was the one commonly used for Linux systems.

5.2 Predictive Performance

In this section, we evaluate the predictive performance of IoTDS. The first part describes the tests setup and the second part the results of the algorithms.

[1] http://docs.python-requests.org/en/master/
[2] http://flask.pocoo.org/
[3] https://joblib.readthedocs.io/en/latest/

5.2.1 Experimental Setup

To evaluate our proposed approach, we divided the capture files in 9 datasets, which are composed of three hours of legitimate traffic and one hour of malicious traffic. Each dataset refers to a different profile and infection type combination as presented in Section 4.4. We organised the evaluation in three phases: finding an optimal time window size, considering that smaller windows will result in faster identification of botnets; discovering the least number of instances needed for training; and optimising the classifier's hyperparameters. Table 5 presents the features and their respective value ranges in the datasets.

Table 5 – Features and their value ranges.

Feature	CPU Utilisation (%)	Memory Utilisation (GB)	Number of Tasks Running	CPU Temperature (C°)
Values	0 - 100	0-1	0-300	0-100

All tests were performed using the holdout method with 50 repetitions to minimise variance in the results. The metrics used to evaluate the proposed system are [40]:

- **Accuracy**:
$$\frac{TP + TN}{TP + FN + FP + TN} \tag{5.1}$$
which describes the overall effectiveness;

- **AUC (Area Under the Curve)**:
$$\frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right) \tag{5.2}$$
the ability of avoiding false classification;

- **Precision**:
$$\frac{TP}{TP + FP} \tag{5.3}$$
the percentage of classified botnets instances that are truly botnets;

- **Recall**:
$$\frac{TP}{TP + FN} \tag{5.4}$$
effectiveness of the approach in identifying botnets;

- **F1-score**:
$$\frac{2}{\frac{1}{recall} + \frac{1}{precision}} \tag{5.5}$$
describes the relation between recall and precision;

- **Specificity**:
$$\frac{TN}{FP + TN} \tag{5.6}$$
which describes the effectiveness in identifying instances that are legitimate.

TP, TN, FP and FN stand for true positives, true negatives, false positives and false negatives, respectively. These metrics were chosen because they are the best ones to evaluate two class problems [40]. We decided not to present the False Positive Rate (FPR), which is found in many network security works, because it is equal to $Specificity - 1$.

5.2.2 Classifiers' Performance

In this section, we discuss the experimental results of the tested algorithms, namely Isolation Forest, Elliptic Envelope, Local Outlier Factor and One-class Support Vector Machine. First, we used the default hyperparameters for the classifiers to analyse the impact on the predictive performance of changing the time window.

Tables 6, 7, 8 and 9 present the average accuracy, precision, recall, F1-score, specificity, and AUC in all datasets for each classifier varying the time window size. It is possible to see that the LOF algorithm had the best results, followed by OSVM, IF and EE considering the F1-score. Although precision is around 70% in the LOF classifier, the other metrics present satisfactory results, considering that no botnet data was used during the training. The best time window size for this algorithm was 30 seconds, followed by 1 second with a small difference between them. Considering the time window with the best performance and a bias towards smaller windows, we chose the 1-second time window. Although it is not the best time window, by choosing the 1-second, botnets would take less time to be detected.

Table 6 – Evaluation of time window size in the LOF in all datasets.

Time window size (s)	Accuracy	Precision	Recall	F1-score	Specificity	AUC
1	0.9233	0.7266	0.9912	0.8385	0.9062	0.9487
5	0.9231	0.7270	0.9893	0.8380	0.9065	0.9479
10	0.9222	0.7248	0.9889	0.8364	0.9055	0.9472
30	0.9248	0.7307	0.9939	0.8419	0.9074	0.9506

Table 7 – Evaluation of time window size in the OSVM in all datasets.

Time window size (s)	Accuracy	Precision	Recall	F1-score	Specificity	AUC
1	0.8985	0.6903	0.8929	0.7778	0.8999	0.8964
5	0.8997	0.6876	0.9183	0.7854	0.8950	0.9067
10	0.8962	0.6848	0.8838	0.7696	0.8993	0.8916
30	0.8938	0.6727	0.8797	0.7578	0.8974	0.8885

After having defined the best time window, we evaluated the amount of legitimate data needed to yield a good predictive performance. First, we fixed the time window to 1-second and again the default parameters for each classifier. Then, samples ranging between 10% and 90% (10-point step) of the legitimate data were extracted to make the training dataset. In other words, at each iteration, for each profile-infection combination, we sampled a percentage of all legitimate data and induced the classifiers on it. After

Table 8 – Evaluation of time window size in the IF in all datasets.

Time window size (s)	Accuracy	Precision	Recall	F1-score	Specificity	AUC
1	0.8366	0.5564	0.9848	0.7100	0.7994	0.8921
5	0.8495	0.5746	0.9879	0.7261	0.8147	0.9013
10	0.8560	0.5915	0.9746	0.7342	0.8261	0.9004
30	0.8287	0.5434	0.9913	0.7010	0.7877	0.8895

Table 9 – Evaluation of time window size in the EE in all datasets.

Time window size (s)	Accuracy	Precision	Recall	F1-score	Specificity	AUC
1	0.8837	0.6599	0.8187	0.7267	0.9001	0.8594
5	0.8979	0.6879	0.8931	0.7759	0.8992	0.8961
10	0.8566	0.6101	0.6883	0.6394	0.8990	0.7937
30	0.8543	0.5542	0.6790	0.5991	0.8984	0.7887

that, the classifiers were tested using the remaining legitimate data and all botnet data. All performance metrics presented in Tables 10, 11, 12 and 13 had very similar results, showing that the classifiers can use the smallest tested fraction of legitimate data to model legitimate behaviour and still reach high predictive performance. This also indicates that all legitimate data are very similar. Thus, the sample size of 0.1 (10%) was chosen.

Table 10 – Evaluation of sample size of the EE in all datasets.

Sample Size	Accuracy	Precision	Recall	F1-score	Specificity	AUC
0.1	0.8846	0.6617	0.8245	0.7303	0.8998	0.8621
0.2	0.8840	0.6605	0.8206	0.7279	0.9000	0.8603
0.3	0.8838	0.6598	0.8193	0.7268	0.9001	0.8597
0.4	0.8837	0.6599	0.8195	0.7269	0.8999	0.8597
0.5	0.8838	0.6601	0.8191	0.7268	0.9001	0.8596
0.6	0.8835	0.6592	0.8175	0.7256	0.9002	0.8588
0.7	0.8835	0.6593	0.8176	0.7257	0.9002	0.8589
0.8	0.8836	0.6592	0.8182	0.7259	0.9001	0.8591
0.9	0.8834	0.6591	0.8175	0.7256	0.9001	0.8588

After selecting the time window and sample size, an optimisation of the classifiers hyperparameters was performed. We used a random search to find the best combination of hyperparameters, as presented in Table 14. All the algorithms were submitted to tests to evaluate their best hyperparameters in the same normalised dataset, except for Isolation Forest, which was not submitted to normalised data because it is a tree ensemble method that can deal with not normalised data. In our evaluation, we used the F1-score to find the best hyperparameters in each algorithm since this metric is a balance between precision and recall. The optimised hyperameters for each algorithm are presented in Table 15. Figures 5, 6, 7 and 8 present the mean performance metrics for each profile and infection type when using a time window of 1 second, and a training sample size corresponding

Table 11 – Evaluation of sample size of the IF in all datasets.

Sample Size	Accuracy	Precision	Recall	F1-score	Specificity	AUC
0.1	0.8372	0.5572	0.9856	0.7109	0.7999	0.8928
0.2	0.8380	0.5583	0.9855	0.7118	0.8009	0.8932
0.3	0.8370	0.5569	0.9856	0.7107	0.7997	0.8927
0.4	0.8384	0.5594	0.9855	0.7126	0.8014	0.8934
0.5	0.8382	0.5590	0.9855	0.7123	0.8012	0.8933
0.6	0.8378	0.5582	0.9856	0.7117	0.8007	0.8931
0.7	0.8378	0.5584	0.9853	0.7118	0.8008	0.8931
0.8	0.8377	0.5581	0.9851	0.7115	0.8007	0.8929
0.9	0.8382	0.5586	0.9855	0.7121	0.8011	0.8933

Table 12 – Evaluation of sample size of the LOF in all datasets.

Sample Size	Accuracy	Precision	Recall	F1-score	Specificity	AUC
0.1	0.9166	0.7119	0.9835	0.8257	0.8998	0.9416
0.2	0.9196	0.7169	0.9915	0.8320	0.9015	0.9465
0.3	0.9210	0.7203	0.9919	0.8345	0.9031	0.9475
0.4	0.9219	0.7227	0.9918	0.8361	0.9043	0.9481
0.5	0.9227	0.7249	0.9915	0.8375	0.9054	0.9484
0.6	0.9235	0.7271	0.9911	0.8388	0.9065	0.9488
0.7	0.9241	0.7289	0.9904	0.8397	0.9074	0.9489
0.8	0.9248	0.7313	0.9889	0.8407	0.9086	0.9488
0.9	0.9255	0.7337	0.9873	0.8418	0.9099	0.9486

Table 13 – Evaluation of sample size of the OSVM in all datasets.

Sample Size	Accuracy	Precision	Recall	F1-score	Specificity	AUC
0.1	0.8978	0.6882	0.8946	0.7770	0.8986	0.8966
0.2	0.8979	0.6890	0.8927	0.7769	0.8992	0.8960
0.3	0.8982	0.6896	0.8928	0.7773	0.8995	0.8962
0.4	0.8984	0.6901	0.8932	0.7778	0.8997	0.8964
0.5	0.8985	0.6903	0.8927	0.7778	0.8999	0.8963
0.6	0.8987	0.6908	0.8931	0.7782	0.9001	0.8966
0.7	0.8985	0.6904	0.8926	0.7778	0.9000	0.8963
0.8	0.8985	0.6904	0.8929	0.7779	0.9000	0.8964
0.9	0.8985	0.6904	0.8928	0.7779	0.8999	0.8964

to 10% of legitimate data. Appendix A includes tables with detailed results for each algorithm.

Since each dataset is composed of 10800 legitimate and 3600 botnet instances, only 1080 (i.e., 10%) legitimate instances were used for training in each technique. The means for specificity and AUC were similarly good for all algorithms. On the other hand,

Table 14 – Hyperparameters used in the methods.

Elliptic Envelope		
Hyperparameter	Range	Number of Samples
contamination	[0, 0.5]	40
Isolation Forest		
Hyperparameter	Range	Number of Samples
n_estimators	{10, 50, 100}	3
contamination	[0, 1)	100
Local Outlier Factor		
Hyperparameter	Range	Number of Samples
n_neighbors	[1, 20]	20
contamination	[0, 0.5)	20
One-class Support Vector Machine (RBF, Linear)		
Hyperparameter	Range	Number of Samples
γ	[0, 1)	20
ν	[0, 1)	20
One-class Support Vector Machine (Poly)		
Hyperparameter	Range	Number of Samples
γ	[0, 1)	20
ν	[0, 1)	20
degree	{2, 3, 4, 5}	4

Table 15 – Optimised Hyperparameters

Elliptic Envelope	
contamination	0.0650
Isolation Forest	
n_estimators	100
contamination	0.0346
Local Outlier Factor	
n_neighbors	12
contamination	0.0091
One-class Support Vector Machine	
γ	0.8941
ν	0.0166
kernel	'rbf'

precision and recall presented some variations for different algorithms, which demands a more detailed analysis and can allow to point out which algorithms achieved better performances.

The first algorithm analysed was the Isolation Forest (Figure 5). Although the classifier presented high values of specificity and AUC, the precision and recall values do not support the use of this method. A clear example of this was the Mirai infection (I2) in the profile ST. In this dataset, the Isolation Forest considers all the instances as legitimate, having a high specificity performance, but both precision and recall presented

values of 0%. The ST profile includes some legitimate interventions on top of the main function of transmitting video, which induced the algorithm to classify the changes caused by Mirai as legitimate. The other datasets presented better results, being the profile SC infected by Mirai the one with the best outcomes.

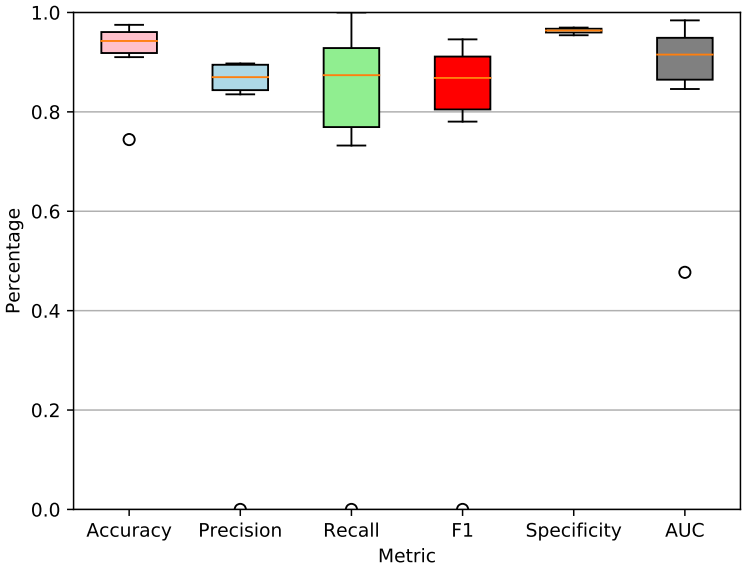

Figure 5 – IF mean results on all datasets with optimised hyperparameters.

The second algorithm analysed is the Elliptic Envelope (Figure 6). This classifier presents a high performance on specificity and AUC, but also the worst results for precision. The worst scenario of this algorithm was the Mirai botnet in the ST profile, and the best results were in the SC profile with Aidra. The high performance on the SC profile can be explained by the lack of additional legitimate interventions from users. All behaviour is only based on the main function, which makes it easier to induce a proper classifier since the normal behaviour is more predictable. The ST profile, on the contrary, has more variations in the normal behaviour, meaning that it is more challenging to the classifier.

Next, Figure 7 presents the results for the One-class Support Vector Machine. It has the second best results of the four algorithms, getting satisfactory results for precision and recall. The scenario with the best results for this classifier was in the SC profile with the Aidra botnet, and the worst was the same profile with the infection type 3 (Mirai, Doflo, Tsunami, Wroba). This worst scenario presented a recall value of 76%, while all other datasets presented better results. This may indicate that, for this specific dataset, the selected hyperparameters were not the best, since they were selected considering all scenarios.

Lastly, Figure 8 presents the results for the LOF algorithm. It had the best average results in all datasets, outperforming the OSVM. The scenario with the best results for

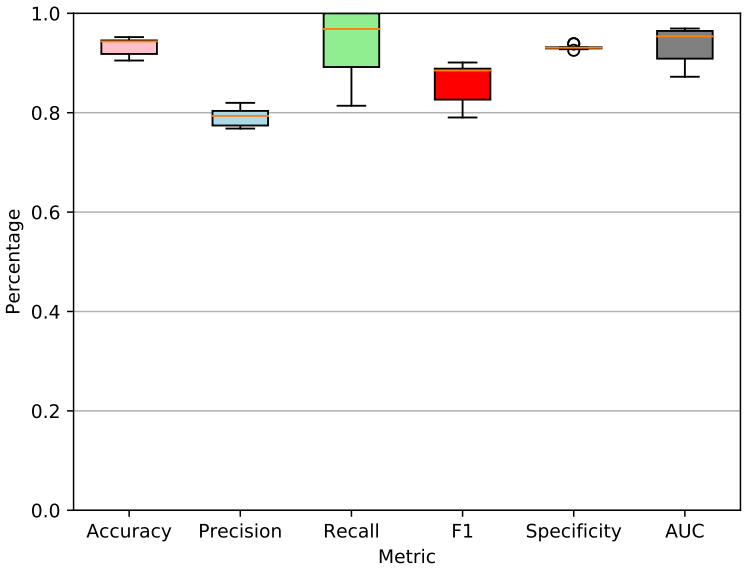

Figure 6 – EE mean results on all datasets with optimised hyperparameters.

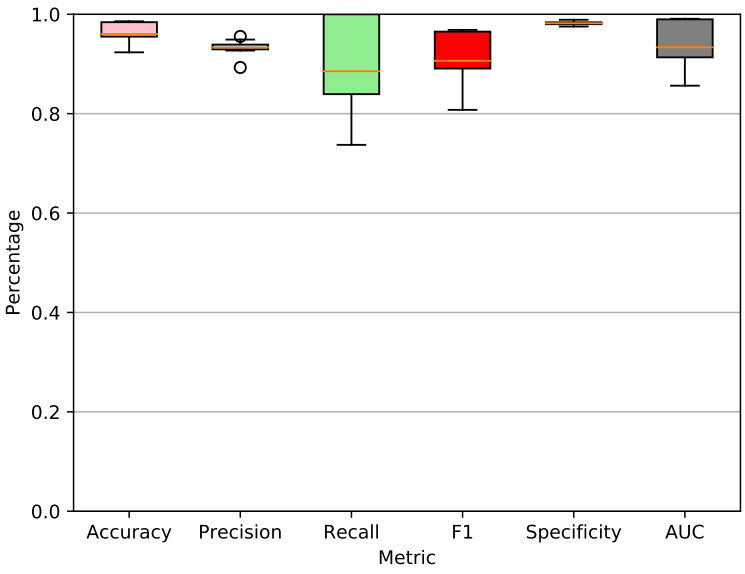

Figure 7 – OSVM mean results on all datasets with optimised hyperparameters.

this algorithm was the infection type 1 (Aidra) on the SC profile with almost 99% of AUC. Also, the precision and recall presented a better performance than the OSVM. The worst results of this algorithm were for the infection type 1 (Bashlite) on the ST profile with a recall of 65%. This may be consequence of the ST profile's peaks of usage, which made the algorithm to classify legitimate variations as botnet behaviour. Nevertheless, the precision remained with a high score.

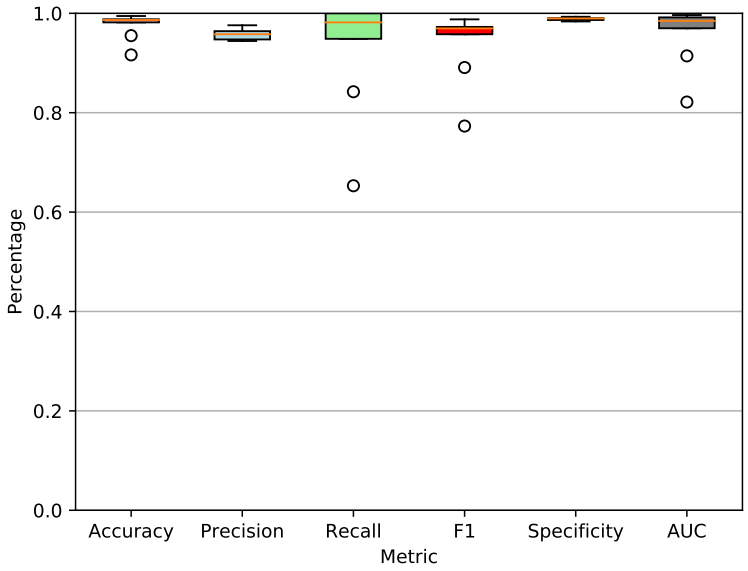

Figure 8 – LOF mean results on all datasets with optimised hyperparameters.

5.3 System impact on its host

After evaluating the predictive performance of our approach, we also need to assess its impact when running in an IoT device. Since we built a host-based approach and IoT devices have limited resources, it is essential that the IoTDS uses the resources from the device in a way not to harm the device's performance. To address this, we computed the device's CPU and memory utilisation, energy consumption, and CPU temperature with and without the IoTDS running. Each monitored parameter has, respectively, the following minimum and maximum values: 0% to 100%, 0 GB to 1 GB, 1 VA to 10 VA and 0°C to 100°C. In Figures 9, 10 and 11, these values, for each IoT device profile, are presented. The blue line represents values computed when the IoTDS was not running, whereas orange lines comprehend to values collected with the IDS running. All the tests were performed with LOF using optimised parameters.

In Figure 9, the MC profile is considered. The IoTDS seems not to have affected the device since the behaviours in the two scenarios are very similar. In this sense the IoTDS did not compromise the device performance. The CPU and memory utilisation, as well as the CPU temperature, behave similarly. The energy consumed in the IDS scenario was slightly higher in average.

Figure 10 presents the results regarding the SC profile. Among the three profiles, this one has the lowest level of resource use under normal conditions, so the impact of the IoTDS is clearer here. The CPU usage increased by 2 percentage points, also followed by the CPU temperature. The memory utilisation increased by 70 MB, but remained

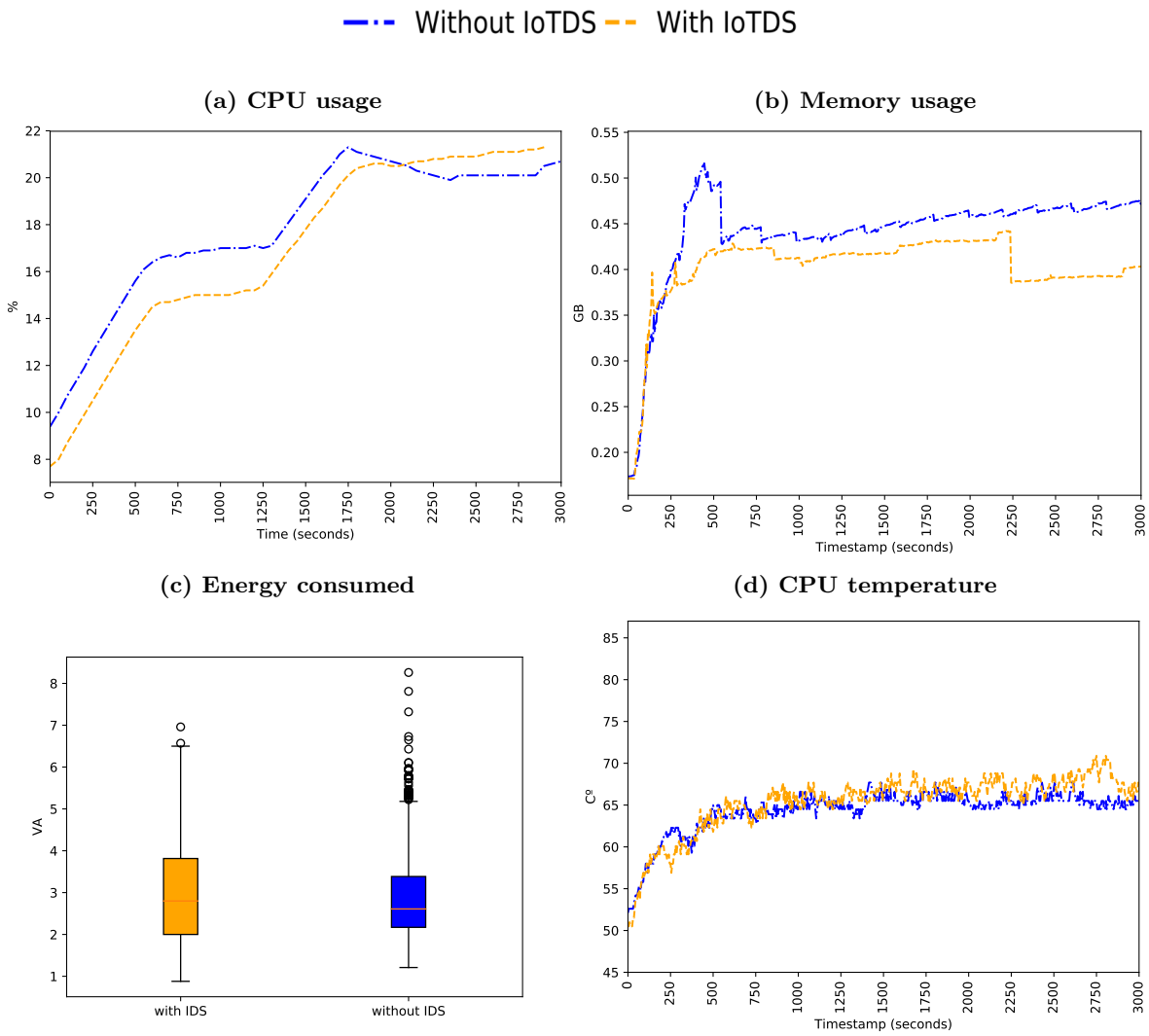

Figure 9 – Behaviour of the MC profile regarding host's resource usage with and without the IoTDS.

constant throughout the time. This profile presented the same increase in the energy consumption as seen in the MC profile. Despite these increases, resources were far from being exhausted, showing that the IoTDS did not compromise the device operation.

Lastly, Figure 11 shows the results for the ST profile. Its behaviour was similar to the SC profile. When this profile was running the IoTDS, the CPU utilisation had a peak of 30%, quickly decreasing later and then, stabilising around 20%. This peak may have been caused by the initialisation of the IoTDS. The energy consumption also increased, having similar results as the MC and SC profiles. In the same manner, as in other profiles, the increase in resource consumption caused by the IoTDS did not seem to impair the device.

For comparison purposes, we also analysed the impact of the IoTDS in a device infected by botnets. Figures 12, 13 and 14 present the CPU and memory utilisation, the energy consumption and the CPU temperature for all the profiles infected by the

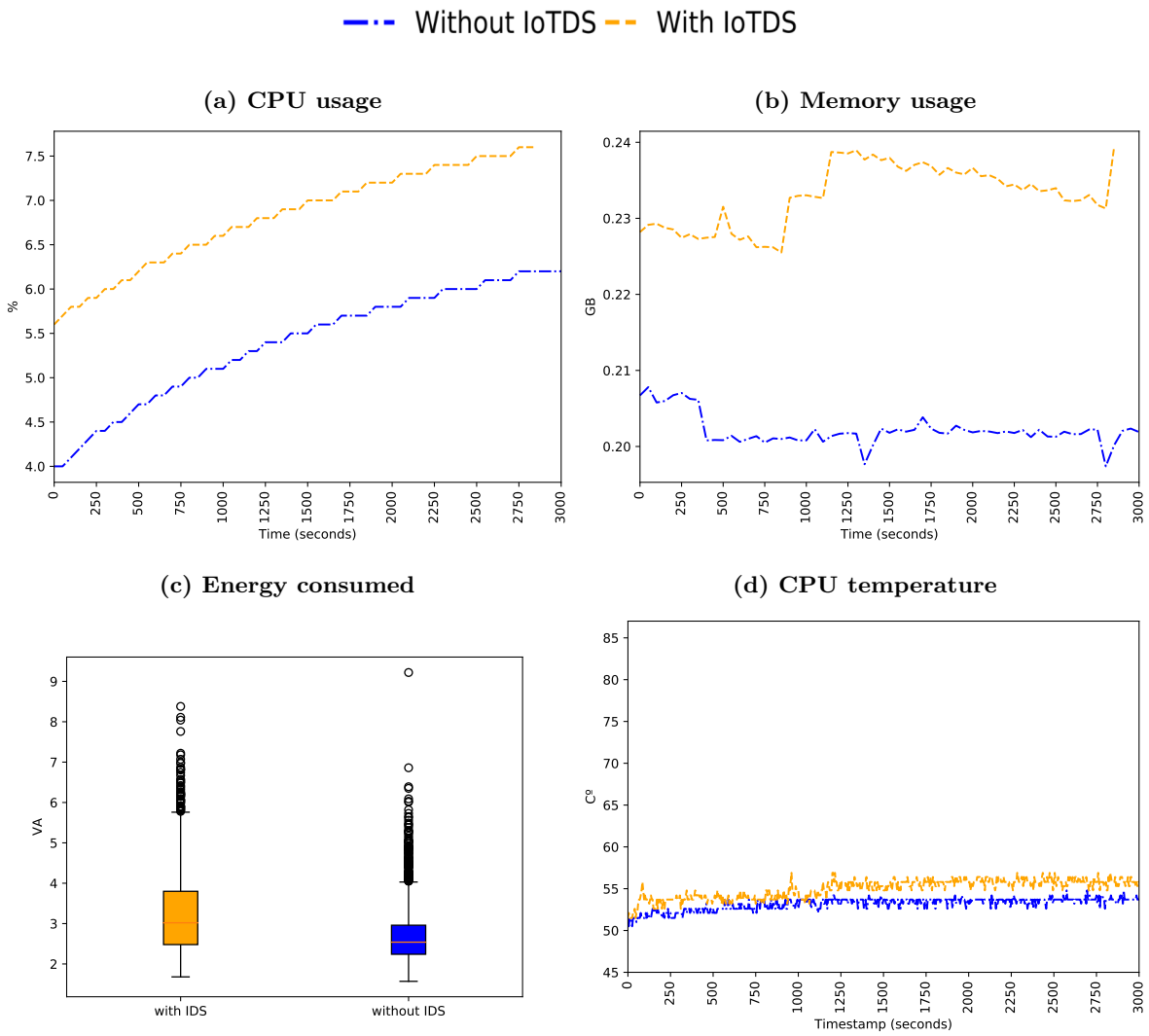

Figure 10 – Behaviour of the SC profile regarding host's resource usage with and without the IoTDS

infection type 3. As shown in the figures, the behaviour was very similar in the cases with and without the IoTDS, as the botnets forced a high consumption of resources on the device. The MC profile showed the highest CPU and memory utilisation, compared to the SC and ST profiles. All resources presented an intense use, regardless of the device was running or not the IoTDS.

In the SC profile, the memory and CPU utilisation, was less intense, representing half of the memory and CPU used by the MC profile. The ST profile shows the same memory utilisation as the SC profile, but a higher CPU utilisation in some moments. In all tests, it is possible to see that the IoTDS impact in the device was almost the same in the infected and non-infected scenarios. This results showed that IoTDS can still be functional even with the device's resources being heavily used by the botnets.

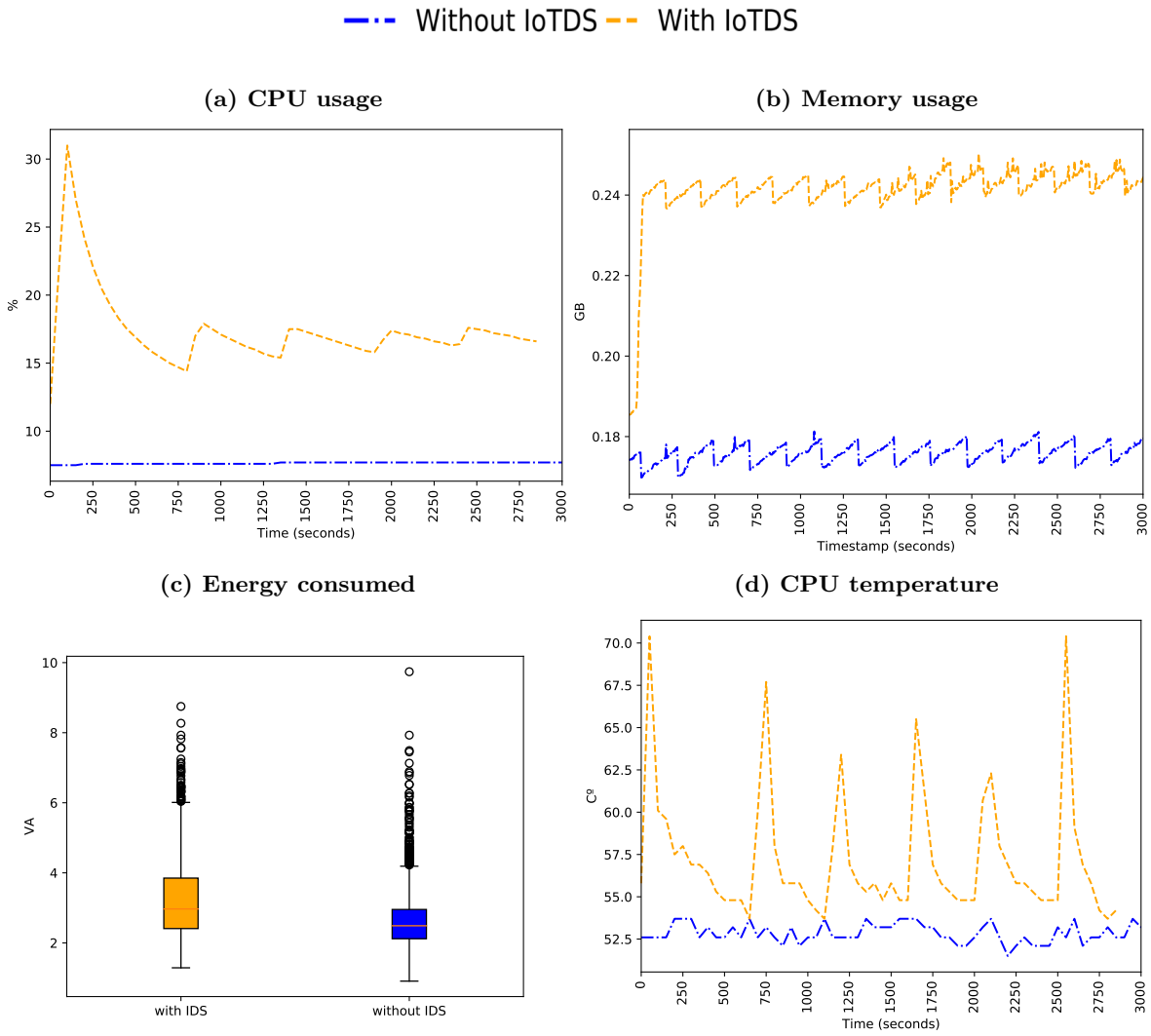

Figure 11 – Behaviour of the ST profile regarding host's resource usage with and without the IoTDS

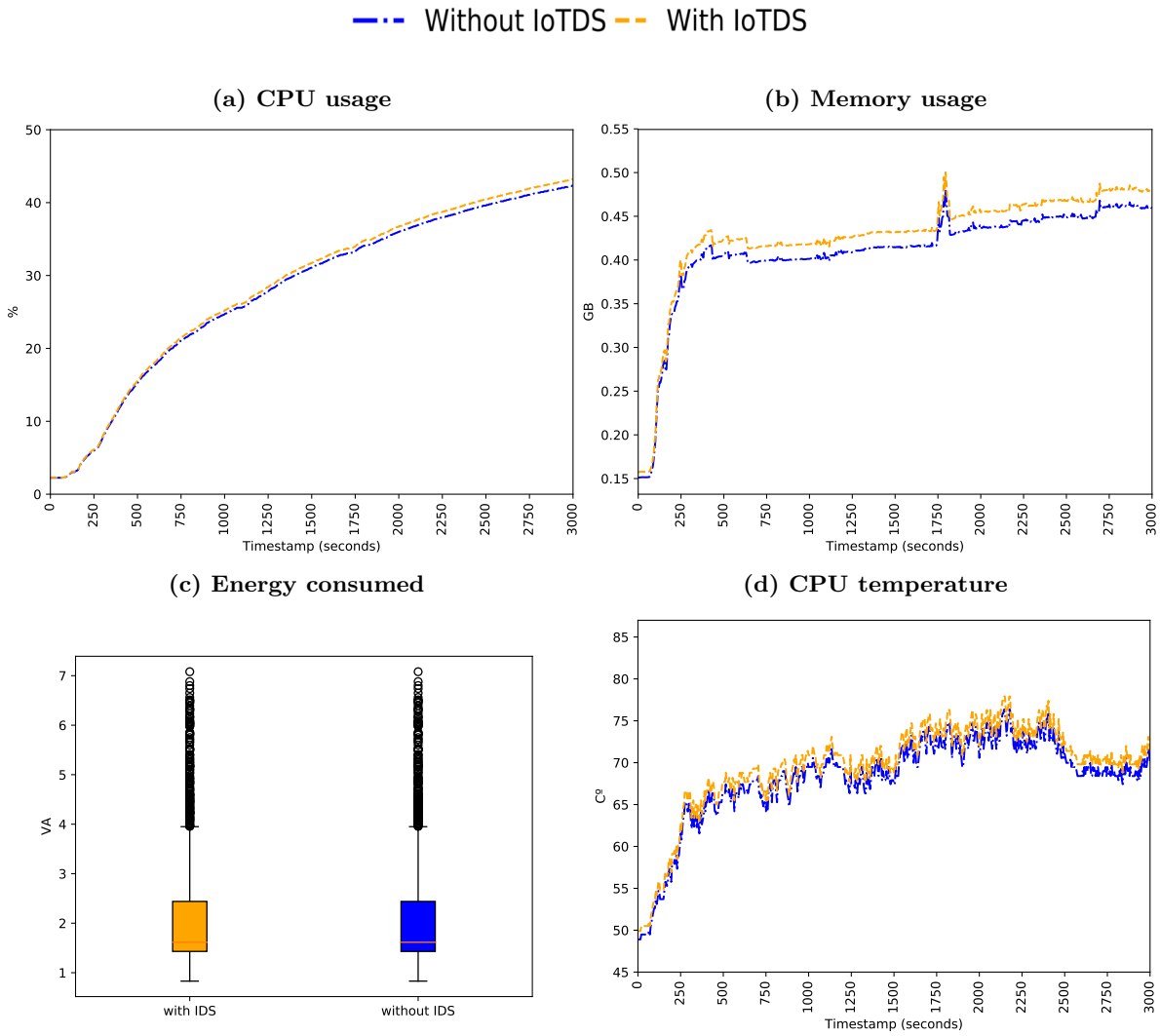

Figure 12 – Behaviour of the MC profile infected with botnets regarding resource usage with and without the IoTDS.

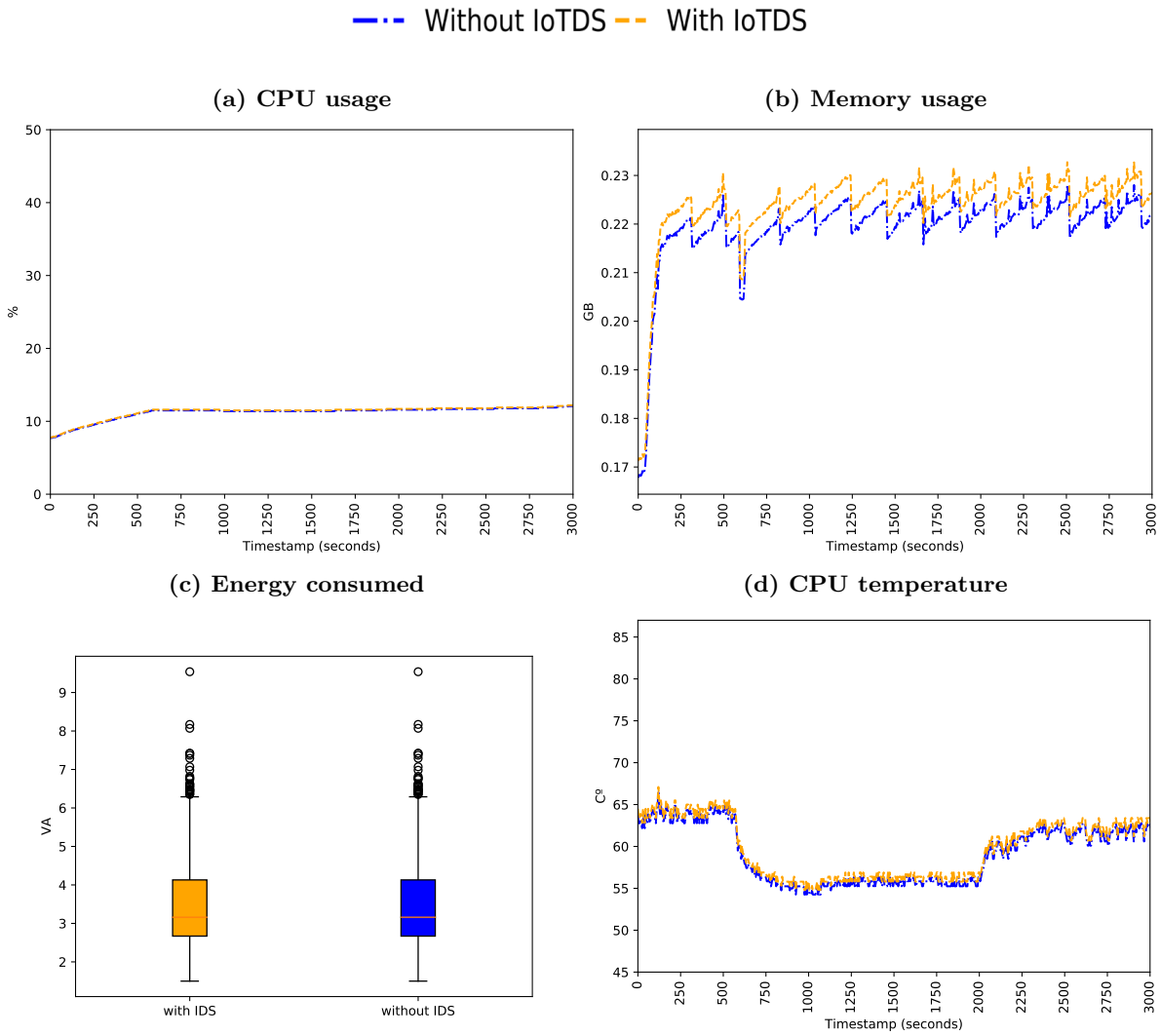

Figure 13 – Behaviour of the SC profile infected with botnets regarding resource usage with and without the IoTDS.

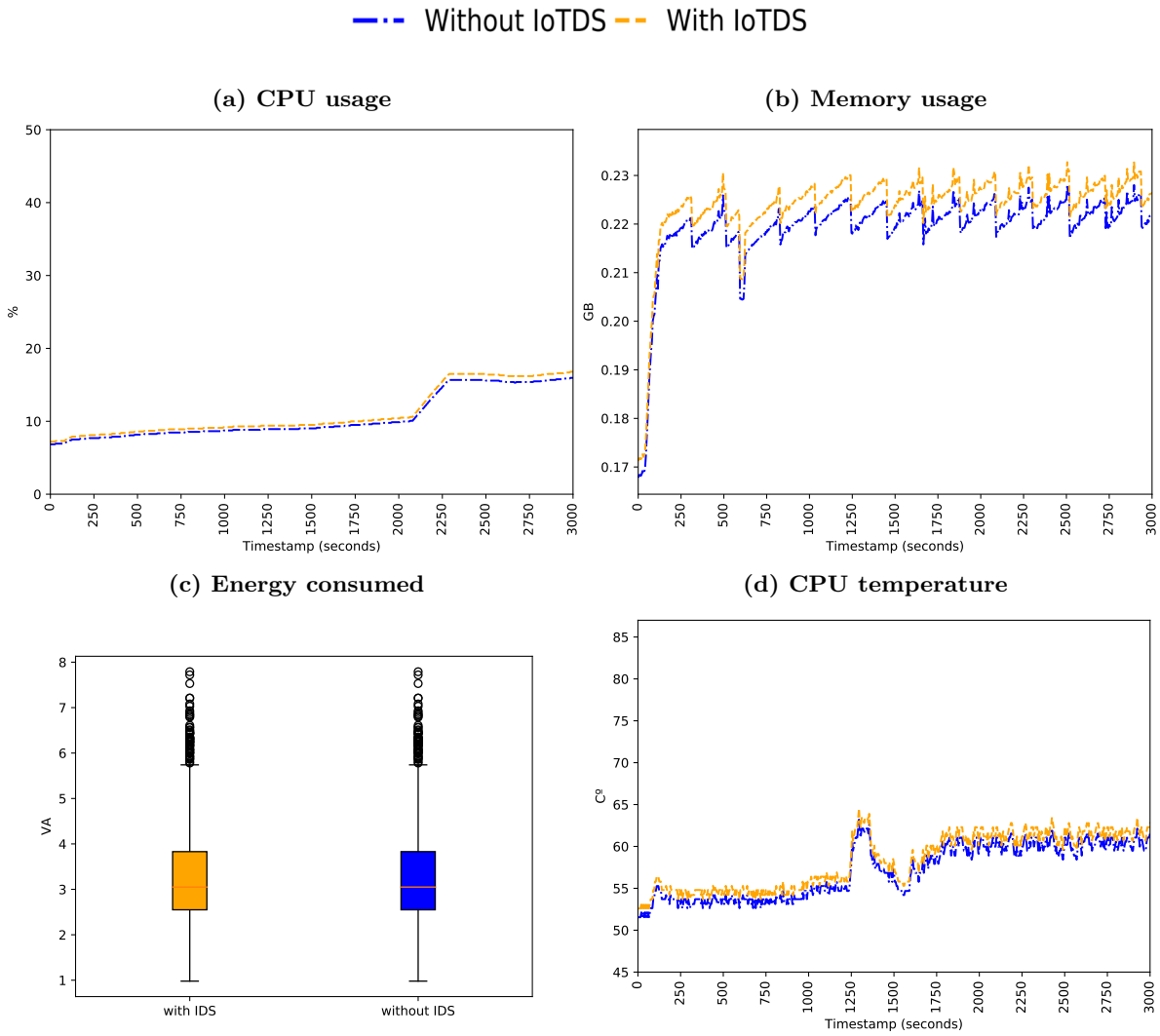

Figure 14 – Behaviour of the ST profile infected with botnets regarding resource usage with and without the IoTDS.

5.4 Discussion

Overall, all algorithms presented high predictive performance, detecting the seven botnets in all profiles. The different legitimate actions the device performed in the proposed profiles seemed to be distinguishable from malicious behaviours by most of the algorithms. All these results were reached using only 18 minutes (1080 instances out of 10800) of data collection to train the models and a time window of 1 second. It means that the system does not require long periods of training and can perform very frequent analysis, on a secondly basis. OSVM and LOF had the best performance out of the four tested classifiers, with LOF holding a slight advantage in most of the datasets.

Analysing the three different profiles tested, the one-class algorithms had a slightly better performance in the SC profile. Due to the peaks of legitimate usage, the ST profile proved to be the most difficult for the algorithms, but they still perform well in this scenario. The most challenging botnets were different for distinct algorithms. For Elliptic Envelope and Local Outlier Factor, Mirai (infection type 2) and Bashlite (infection type 1) were the hardest ones. For the Isolation Forest and One-class SVM, the infection type 3 (Mirai, Doflo, Tsunami and Wroba) was the most difficult.

The used host-based features showed to be accurate indicators of malware presence in the device. It is noteworthy to say that the adoption of host based data does not rule out the possibility of using other types of data. Using host-based data in conjunction with network-based data, for example, can provide even more security for the entire system, including hosts and network. The impact of the IoTDS in the performance of the device was also considered. Our tests showed that although there was some increase in resource consumption when the IoTDS was running, this was far from hindering the capabilities of the device.

Considering the memory and CPU utilisation of the Raspberry Pi during the tests, it is possible to say that the IoTDS could be a solution for multiple IoT devices found in today's households. Examples of these IoT devices are: Odroid-XU4[4], which has a 2GB RAM and a Octa-Core processor, Roku TV[5], which has a 512MB-1GB RAM in the 4k version with a ARM Quad-Core processor, and smart TVs such as the TCL L40S4900FS, which has a Quad-Core processor and a 1GB RAM.

[4] https://www.hardkernel.com/
[5] https://www.roku.com/en-gb/

6 CONCLUSION

The growing number of IoT devices brings new threats to network security since these devices are usually more insecure than desktop computers. Botnets are a significant threat in this scenario, using multiple devices to perform synchronised malicious actions. To tackle this problem, new approaches to protect those devices and their networks are needed. One possible solution is to develop IDS to detect infected devices and then take the necessary mitigation measures. Although IDS for IoT devices have been developed by the research community, they still present some issues, such as focusing only on specific network protocols (e.g., 6LoWPAN in Contiki operating system), relying on labelled data, exploring only network data and not being validated with newer botnets. Also, researchers do not have access to a large number of IoT datasets to evaluate the performance of their approaches.

This work proposed a solution for these problems with the development of botnet detection approach for IoT devices, as well as a dataset to evaluate the proposed system. Most IoT devices run simple well-defined tasks, presenting a systematic behaviour as long as they are not compromised. Thus, our idea was to use this well-defined behaviour to create a model based on legitimate host-based data from IoT devices and compare the model with new data to check for anomalies. One-class classifiers were chosen to underpin this solution.

Our approach, named IoTDS, was tested with four one-class algorithms, namely Elliptic Envelope, Isolation Forest, One-class Support Vector Machine and Local Outlier Factor. IoTDS is divided into two elements: the IoTDS Agent, installed in the IoT device, and the IoTDS Management Console, installed in a separated server. Firstly, the Agent is responsible for collecting the host's memory and CPU utilisation, number of running tasks, and CPU temperature, for a period of time and sending this data to the Mangement Console. The Management Console then use this data to create a one-class classifier and send back to the Agent the model created. In the next phase, the Agent uses the classifier to analyse the device behaviour and, in case of anomaly, sends an alert to the Management Console. In this proposed architecture, the burden of inducing new classifiers is transferred to a dedicated server, leaving the IoT devices free from running this costly task. In addition to data collection, the only responsibility of IoTDS Agents is to analyse new data using the classifier induced by the Management Console.

To create the dataset to evaluate the proposed system, we developed an IoT experimental environment using a Raspberry Pi. This environment is an intranet composed of a Raspberry Pi connected wirelessly, four desktop computers and a wireless router. Three operational profiles were defined for the Raspberry Pi, aiming to emulate typical domestic

IoT devices, such as surveillance cameras and multimedia centres. The four desktop computers acted as a gateway, a DNS server which consumes video from the Raspberry Pi, a device administrator, a web server and an attacker's host. For each profile, we collected data from the device while was not infected and in three different infection scenarios. In addition to the data used in this work, the dataset includes data related to the network traffic on the Raspberry Pi. The dataset was not limited only to the data used in this work because the objective was to make publicly available to support other researchers working on the same topic.

To evaluate IoTDS, we divided the files of the experimental environment in 9 datasets composed of three hours of legitimate traffic and one hour of malicious traffic. We organised the tests to evaluate the system's predictive performance and its impact on the host. First, we analysed the impact of the time window and the amount of legitimate data needed to train the model in the predictive performance. We tested four time windows and nine training sample sizes with the algorithms using their default hyperparameters. Our tests showed that a sample size of 18 minutes for training and a time window of 1 second would be able to produce results as good as those obtained with larger training samples and time windows.

After selecting the time window and the training sample size, we optimised the hyperparameters of all tested algorithms. In our tests, the best algorithms in all datasets were OSVM and LOF, with the LOF algorithm having a slight advantage with a mean F1-score of 94%. We also evaluated the impact of running the IoTDS in the three different IoT device's profiles. These tests showed that the approach increases the CPU, memory and energy utilisation of the device, but we did not detect any problems for the device's operation.

As future work, we intend to evaluate the proposed approach in different devices and IoT botnets, including other host-based features (e.g., syscalls) and testing other ways of mitigating botnets, such as blocking IP addresses and changing IPs using software-defined networking routers in conjunction with IoTDS.

BIBLIOGRAPHY

[1] WHITMORE, A.; AGARWAL, A.; Da Xu, L. The Internet of Things—A survey of topics and trends. *Information Systems Frontiers*, v. 17, n. 2, p. 261–274, 2015. ISSN 15729419.

[2] ANGRISHI, K. Turning Internet of Things(IoT) into Internet of Vulnerabilities (IoV) : IoT Botnets. *arXiv preprint arXiv:1702.03681*, p. 1–17, 2017. Disponível em: <http://arxiv.org/abs/1702.03681>.

[3] KOLIAS, C. et al. DDoS in the IoT: Mirai and Other Botnets. *Computer*, IEEE, v. 50, n. 7, p. 80–84, 2017.

[4] MANSFIELD-DEVINE, S. DDoS goes mainstream: how headline-grabbing attacks could make this threat an organisation's biggest nightmare. *Network Security*, v. 2016, n. 11, p. 7 – 13, 2016. ISSN 1353-4858. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1353485816301040>.

[5] BERTINO, E.; ISLAM, N. Botnets and Internet of Things Security. *Computer*, v. 50, n. 2, p. 76–79, 2017. ISSN 0018-9162. Disponível em: <http://ieeexplore.ieee.org/document/7842850/>.

[6] RAZA, S.; WALLGREN, L.; VOIGT, T. SVELTE: Real-time intrusion detection in the Internet of Things. *Ad hoc networks*, Elsevier, v. 11, n. 8, p. 2661–2674, 2013.

[7] AMARAL, J. P. et al. Policy and network-based intrusion detection system for IPv6-enabled wireless sensor networks. In: IEEE. *Communications (ICC), 2014 IEEE International Conference on*. [S.l.], 2014. p. 1796–1801.

[8] GARCÍA, S.; ZUNINO, A.; CAMPO, M. Survey on network-based botnet detection methods. *Security and Communication Networks*, Wiley Online Library, v. 7, n. 5, p. 878–903, 2014.

[9] ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: A survey. *Computer networks*, Elsevier, v. 54, n. 15, p. 2787–2805, 2010.

[10] GUBBI, J. et al. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, Elsevier B.V., v. 29, n. 7, p. 1645–1660, 2013. ISSN 0167739X. Disponível em: <http://dx.doi.org/10.1016/j.future.2013.01.010>.

[11] HABIBI, J. et al. Heimdall: Mitigating the Internet of Insecure Things. *IEEE Internet of Things Journal*, IEEE, 2017.

[12] ZARPELÃO, B. B. et al. A survey of intrusion detection in Internet of Things. *Journal of Network and Computer Applications*, Elsevier Ltd, v. 84, n. September 2016, p. 25–37, 2017. ISSN 10848045. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S1084804517300802>.

[13] BORGIA, E. The internet of things vision: Key features, applications and open issues. *Computer Communications*, Elsevier, v. 54, p. 1–31, 2014.

[14] SICARI, S. et al. Security, privacy and trust in internet of things: The road ahead. *Computer Networks*, v. 76, p. 146 – 164, 2015. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128614003971>.

[15] ZARGAR, S. T.; JOSHI, J.; TIPPER, D. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys and Tutorials*, v. 15, n. 4, p. 2046–2069, 2013. ISSN 1553877X.

[16] PENG, T.; LECKIE, C.; RAMAMOHANARAO, K. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys*, v. 39, n. 1, p. 3–es, 2007. ISSN 03600300. Disponível em: <http://portal.acm.org/citation.cfm?doid=1216370.1216373>.

[17] SILVA, S. S. C. et al. Botnets: A survey. *Computer Networks*, v. 57, n. 2, p. 378–403, 2013. ISSN 13891286.

[18] COSTA, V. G. Turrisi da et al. Detecting Mobile Botnets Through Machine Learning and System Calls Analysis. *Proceedings of the 2017 IEEE International Conference on Communications (ICC)*, p. 917–922, 2017.

[19] STAVROU, A.; VOAS, J.; FELLOW, I. DDoS in the IoT. *Computer*, v. 50, p. 80–84, 2017. ISSN 0018-9162.

[20] KADIR, A. F. A.; STAKHANOVA, N.; GHORBANI, A. A. Android botnets: What urls are telling us. In: QIU, M. et al. (Ed.). *Network and System Security*. Cham: Springer International Publishing, 2015. p. 78–91. ISBN 978-3-319-25645-0.

[21] NIGAM, R. A timeline of mobile botnets. *Virus Bulletin, March*, 2015.

[22] LIAO, H.-J. et al. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, Elsevier, v. 36, n. 1, p. 16–24, 2013.

[23] BUCZAK, A. L.; GUVEN, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, IEEE, v. 18, n. 2, p. 1153–1176, 2016.

[24] ABDUVALIYEV, A. et al. On the vital areas of intrusion detection systems in wireless sensor networks. *IEEE Communications Surveys & Tutorials*, IEEE, v. 15, n. 3, p. 1223–1237, 2013.

[25] RESENDE, P. A. A.; DRUMMOND, A. C. A survey of random forest based methods for intrusion detection systems. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 51, n. 3, p. 48:1–48:36, maio 2018. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/3178582>.

[26] SHAMS, E. A.; RIZANER, A. A novel support vector machine based intrusion detection system for mobile ad hoc networks. *Wireless Networks*, v. 24, n. 5, p. 1821–1829, Jul 2018. ISSN 1572-8196. Disponível em: <https://doi.org/10.1007/s11276-016-1439-0>.

[27] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

[28] SHONE, N. et al. A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, IEEE, v. 2, n. 1, p. 41–50, 2018.

[29] COSTA, V. G. Turrisi da et al. Online detection of botnets on network flows using stream mining. In: SBC. *Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.], 2018.

[30] CARVALHO, A. et al. Inteligência artificial–uma abordagem de aprendizado de máquina. *Rio de Janeiro: LTC*, 2011.

[31] MITCHELL, T. *Machine Learning*. McGraw-Hill Education, 1997. (McGraw-Hill international editions - computer science series). ISBN 9780070428072. Disponível em: <https://books.google.com.br/books?id=xOGAngEACAAJ>.

[32] FLACH, P. *The Art and Science of Algorithms that Make Sense of Data*. [S.l.]: Cambridge University Press, 2012.

[33] MAZHELIS, O. One-class classifiers: a review and analysis of suitability in the context of mobile-masquerader detection. *South African Computer Journal*, Sabinet, v. 2006, n. 36, p. 29–48, 2006.

[34] KHAN, S. S.; MADDEN, M. G. A survey of recent trends in one class classification. In: SPRINGER. *Irish Conference on Artificial Intelligence and Cognitive Science*. [S.l.], 2009. p. 188–197.

[35] SHIN, H. J.; EOM, D.-H.; KIM, S.-S. One-class support vector machines—an application in machine fault detection and classification. *Computers & Industrial Engineering*, Elsevier, v. 48, n. 2, p. 395–408, 2005.

[36] ROUSSEEUW, P. J.; DRIESSEN, K. V. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, Taylor & Francis Group, v. 41, n. 3, p. 212–223, 1999.

[37] HOYLE, B. et al. Anomaly detection for machine learning redshifts applied to sdss galaxies. *Monthly Notices of the Royal Astronomical Society*, The Royal Astronomical Society, v. 452, n. 4, p. 4183–4194, 2015.

[38] LIU, F. T.; TING, K. M.; ZHOU, Z.-H. Isolation forest. In: IEEE. *2008 Eighth IEEE International Conference on Data Mining*. [S.l.], 2008. p. 413–422.

[39] BREUNIG, M. M. et al. Lof: identifying density-based local outliers. In: ACM. *ACM sigmod record*. [S.l.], 2000. v. 29, n. 2, p. 93–104.

[40] SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, Elsevier, v. 45, n. 4, p. 427–437, 2009.

[41] MANEVITZ, L. M.; YOUSEF, M. One-class svms for document classification. *Journal of machine Learning research*, v. 2, n. Dec, p. 139–154, 2001.

[42] GUERBAI, Y.; CHIBANI, Y.; HADJADJI, B. The effective use of the one-class svm classifier for handwritten signature verification based on writer-independent parameters. *Pattern Recognition*, Elsevier, v. 48, n. 1, p. 103–113, 2015.

[43] VAPNIK, V. *The Nature of Statistical Learning Theory*. [S.l.]: Springer, 1995.

[44] MEIDAN, Y. et al. N-baiot: Network-based detection of iot botnet attacks using deep autoencoders. *arXiv preprint arXiv:1805.03409*, 2018.

[45] AN, N. et al. Behavioral anomaly detection of malware on home routers. In: IEEE. *Malicious and Unwanted Software (MALWARE), 2017 12th International Conference on*. [S.l.], 2017. p. 47–54.

[46] BAY, S. D. et al. The UCI KDD Archive of Large Data Sets for Data Mining Research and Experimentation. *SIGKDD Explor. Newsl.*, ACM, New York, NY, USA, v. 2, n. 2, p. 81–85, dez. 2000. ISSN 1931-0145. Disponível em: <http://doi.acm.org/10.1145/380995.381030>.

[47] SHIRAVI, A. et al. Toward Developing a Systematic Approach to Generate Benchmark Datasets for Intrusion Detection. *Comput. Secur.*, Elsevier Advanced Technology Publications, Oxford, UK, UK, v. 31, n. 3, p. 357–374, maio 2012. ISSN 0167-4048. Disponível em: <http://dx.doi.org/10.1016/j.cose.2011.12.012>.

[48] GARCIA, S. *Identifying, Modeling and Detecting Botnet Behaviors in the Network*. Tese (Doutorado) — Universidad Nacional del Centro de la Provincia de Buenos Aires, 2014.

[49] THOMAS, C.; SHARMA, V.; BALAKRISHNAN, N. Usefulness of DARPA dataset for Intrusion Detection System Evaluation. *SPIE 6973, Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security*, v. 6973, p. 69730G–69730G–8, 2008. ISSN 0277786X.

[50] STALLINGS, W. *Cryptography and network security: principles and practice*. [S.l.]: Pearson Upper Saddle River, 2017.

[51] SUMMERVILLE, D. H.; ZACH, K. M.; CHEN, Y. Ultra-lightweight deep packet anomaly detection for Internet of Things devices. In: IEEE. *Computing and Communications Conference (IPCCC), 2015 IEEE 34th International Performance*. [S.l.], 2015. p. 1–8.

[52] NOBAKHT, M.; SIVARAMAN, V.; BORELI, R. A Host-Based Intrusion Detection and Mitigation Framework for Smart Home IoT Using OpenFlow. In: IEEE. *Availability, Reliability and Security (ARES), 2016 11th International Conference on*. [S.l.], 2016. p. 147–156.

[53] OH, D.; KIM, D.; RO, W. W. A malicious pattern detection engine for embedded security systems in the Internet of Things. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 14, n. 12, p. 24188–24211, 2014.

[54] SFORZIN, A. et al. RPiDS: Raspberry Pi IDS—A Fruitful Intrusion Detection System for IoT. In: IEEE. *UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld, 2016 Intl IEEE Conferences*. [S.l.], 2016. p. 440–448.

[55] NAYYAR, A.; PURI, V. Raspberry Pi-A Small, Powerful, Cost Effective and Efficient Form Factor Computer: A Review. *International Journal of Advanced Research in Computer Science and Software Engineering*, v. 5, n. 12, p. 720–737, 2015. Disponível em: <http://www.ijarcsse.com/docs/papers/Volume{_}5/12{_}December2015/V5I12-03>.

[56] PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011.

Appendix

APPENDIX A – DETAILED RESULTS OF THE ALGORITHMS

In this chapter is presented the detailed results of the algorithms tested in this work:

Table 16 – Results of an optimised Isolation Forest for all datasets.

Dataset	Accuracy	Precision	Recall	F1	Specificity	AUC
MC_I1	0.9463	0.8782	0.8750	0.8766	0.9661	0.9206
MC_I2	0.9217	0.8578	0.7694	0.8112	0.9643	0.8668
MC_I3	0.9460	0.8711	0.8841	0.8775	0.9634	0.9237
ST_I1	0.9349	0.8442	0.8605	0.8523	0.9556	0.9081
ST_I2	0.7472	0.0000	0.0000	0.0000	0.9580	0.4790
ST_I3	0.9123	0.8334	0.7472	0.7880	0.9584	0.8528
SC_I1	0.9737	0.8922	1.0000	0.9430	0.9664	0.9832
SC_I2	0.9784	0.9095	1.0000	0.9526	0.9724	0.9862
SC_I3	0.9429	0.8790	0.8554	0.8671	0.9672	0.9113
mean	0.9226	0.7739	0.7769	0.7743	0.9635	0.8702
std	0.0691	0.2912	0.3038	0.2952	0.0053	0.1533

Table 17 – Results of an optimised Elliptic Envelope for all datasets.

Dataset	Accuracy	Precision	Recall	F1	Specificity	AUC
MC_I1	0.9182	0.7697	0.8919	0.8263	0.9256	0.9087
MC_I2	0.9465	0.8036	1.0000	0.8911	0.9316	0.9658
MC_I3	0.9295	0.7906	0.9222	0.8513	0.9316	0.9269
ST_I1	0.9158	0.7742	0.8672	0.8181	0.9294	0.8983
ST_I2	0.9050	0.7681	0.8140	0.7904	0.9307	0.8723
ST_I3	0.9454	0.8006	0.9983	0.8886	0.9307	0.9645
SC_I1	0.9521	0.8198	1.0000	0.9010	0.9388	0.9694
SC_I2	0.9434	0.7936	1.0000	0.8849	0.9277	0.9639
SC_I3	0.9453	0.8151	0.9685	0.8852	0.9388	0.9536
mean	0.9335	0.7928	0.9402	0.8597	0.9316	0.9359
std	0.0168	0.0190	0.0697	0.0396	0.0045	0.0357

Table 18 – Results of an optimised OSVM for all datasets.

Dataset	Accuracy	Precision	Recall	F1	Specificity	AUC
MC_I1	0.9600	0.9270	0.8867	0.9064	0.9805	0.9336
MC_I2	0.9550	0.9491	0.8393	0.8908	0.9874	0.9133
MC_I3	0.9576	0.9554	0.8457	0.8972	0.9889	0.9173
ST_I1	0.9619	0.9365	0.8854	0.9103	0.9832	0.9343
ST_I2	0.9841	0.9325	1.0000	0.9651	0.9796	0.9898
ST_I3	0.9234	0.8929	0.7373	0.8077	0.9754	0.8563
SC_I1	0.9858	0.9390	1.0000	0.9685	0.9819	0.9910
SC_I2	0.9844	0.9331	1.0000	0.9654	0.9801	0.9900
SC_I3	0.9366	0.9294	0.7676	0.8408	0.9838	0.8757
mean	0.9610	0.9328	0.8847	0.9058	0.9823	0.9335
std	0.0217	0.0176	0.0993	0.0561	0.0041	0.0495

Table 19 – Results of an optimised LOF for all datasets.

Dataset	Accuracy	Precision	Recall	F1	Specificity	AUC
MC_I1	0.9817	0.9641	0.9518	0.9579	0.9901	0.9709
MC_I2	0.9879	0.9637	0.9818	0.9726	0.9896	0.9857
MC_I3	0.9819	0.9679	0.9487	0.9582	0.9912	0.9699
ST_I1	0.9163	0.9474	0.6530	0.7731	0.9899	0.8214
ST_I2	0.9946	0.9759	1.0000	0.9878	0.9930	0.9965
ST_I3	0.9866	0.9578	0.9817	0.9696	0.9879	0.9848
SC_I1	0.9893	0.9533	1.0000	0.9761	0.9864	0.9932
SC_I2	0.9872	0.9443	1.0000	0.9713	0.9836	0.9918
SC_I3	0.9551	0.9457	0.8422	0.8909	0.9865	0.9144
mean	0.9756	0.9578	0.9288	0.9397	0.9887	0.9587
std	0.0249	0.0110	0.1147	0.0684	0.0029	0.0572

PAPERS PUBLISHED BY THE AUTHOR

Papers published by the author during the program.

1. Vitor Hugo Bezerra, Victor G. Turrisi da Costa, Ricardo Augusto Martins, Sylvio Barbon Junior, Rodrigo Sanches Miani, Bruno Bogaz Zarpelão, **Providing IoT host-based datasets for intrusion detection research**, SBSeg (Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais), 2018, SBC, (Qualis CC 2017, B3)

2. Vitor Hugo Bezerra, Victor G. Turrisi da Costa, Sylvio Barbon Junior, Rodrigo Sanches Miani, Bruno Bogaz Zarpelão, **One-class Classification to Detect Botnets in IoT devices**, SBSeg (Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais), 2018, SBC, (Qualis CC 2017, B3)