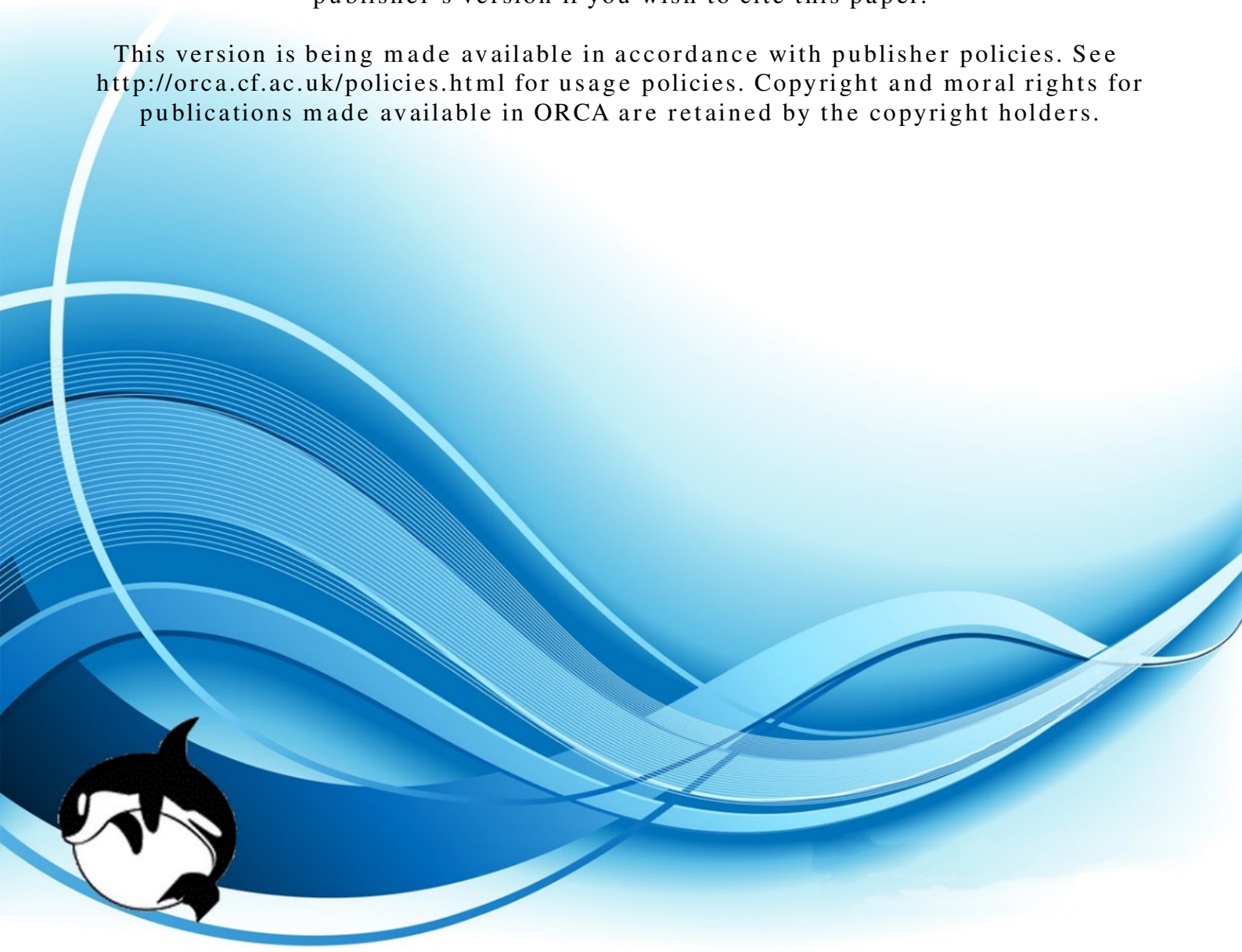

EclipseIoT: A Secure and Adaptive Hub for the Internet of Things

Eirini Anthi*, Shazaib Ahmad, Omer Rana, George Theodorakopoulos, Pete Burnap
Cardiff University, School of Computer Science & Informatics, 5 The Parade, Roath, Cardiff, CF24 3AA

Abstract

With the proliferation in the quantity and types of devices that may be included in an Internet of Things (IoT) ecosystem, particularly in the context of a smart home, it is essential to provide mechanisms to deal with the heterogeneity which such devices encompass. Variations can occur in data formats, frequency of operation, or type of communication protocols supported. The ability to support integration between sensors using a "hub" has become central to address many of these issues. The implementation of such a hub can provide both the ability to act as an aggregator for various sensors, and also limit an attacker's visibility into locally provisioned sensing capability. This paper introduces EclipseIoT, an adaptive hub which uses dynamically loadable add-on modules to communicate with diverse IoT devices, provides policy-based access control, limits exposure of local IoT devices through *cloaking*, and offers a *canary-function* based capability to monitor attack behaviours. Its architecture and implementation are discussed, along with its use within a smart home testbed consisting of commercially available devices such as Phillips Hue Bridge, Samsung Smart Things Hub, TP-Link Smart Plug, and TP-Link Smart Camera. The effectiveness of EclipseIoT is further evaluated by simulating various attacks such as Address Resolution Protocol (ARP) spoofing, Media Access Control (MAC) address spoofing, Man-In-The-Middle (MITM), port scanning, capturing handshakes, sniffing, and Denial of Service (DoS). It is demonstrated that direct attacks upon EclipseIoT components are mitigated due to the security techniques being used.

Keywords: Internet of Things (IoT), Networking, Security, Framework, Hub

*Corresponding author
Email address: anthies@cardiff.ac.uk (Eirini Anthi)

1. Introduction

The Internet of Things (IoT) is the system of interconnected electronic devices embedded with software, sensors, actuators, and network connectivity which enable them to connect and exchange data [1]. IoT devices such as smart and wearable devices, home appliances, and alarm and camera systems provide various functionalities which automate and support our daily activities and needs. For instance, smart fitness trackers such as Fitbit allow users to track their physical movements in order to measure and set personal fitness goals. However, IoT devices are not only used in domestic environments, but are also employed in larger networks such as Critical National Infrastructures (CNI). These include concepts that may be necessary for a country to function and upon which our daily life depends on, such as smart cities, intelligent transport, smart grids, and our health care systems.

The proliferation of IoT devices in the past decade is demonstrated by how prevalent they have become in our lives. Gartner [2] predicts that by 2020, there will be 20.8 billion IoT devices connected around the world, overtaking the number of personal computers and smartphones combined [3]. These devices are pervasive and have access to sensitive data such as location, usernames, passwords, etc. [4]. Although IoT is considered as being the next 'Industrial Revolution', which is shifting how us as individuals, economic entities, and governmental organisations interact with the physical world, such technologies come with enormous security flaws [3, 5, 6, 7]. For instance, a recent study by Hewlett Packard Enterprise [4] investigated the security of 10 of the most popular IoT devices. They discovered that each device had a recklessly high number of security vulnerabilities, each suffering from, on average, 25 issues, including Heartbleed, Denial of Service (DoS), weak passwords, and cross-site scripting. Other surveys have also found similar limitations, such as the OWASP Top 10 IoT Vulnerabilities [8]. Moreover, IoT devices have recently been employed as part of botnets, such as *Mirai*, and have launched several of the largest Distributed Denial of Service (DDoS) and spam attacks [9]. Given that IoT devices suffer from the aforementioned vulnerabilities and that they are often deeply embedded in networks, they are considered to be the 'weakest link' for breaking into a secure infrastructure and have become attractive targets for attacks. As IoT devices have a direct impact on our lives, security and privacy considerations must become a higher priority. Therefore, the need to develop a robust and well-defined security infrastructure, with new systems and protocols that can limit the possible threats related to IoT devices, is greater than ever [10].

In comparison to traditional IT systems, various factors in the IoT ecosystem suggest that implementing such a security infrastructure may be challenging [11]. Unlike traditional systems, IoT devices are small and have limited computational capabilities. As a result, the security mechanisms that these devices can support are weaker. Additionally, as these smart devices may handle sensitive data, for example regarding our health and behaviours, it is possible for such information to be leaked, breaching their security. Finally, when considering the implementation of IoT security mechanisms, one of the most challenging aspect is the heterogeneity across such devices [11] in terms of their hardware, software, and communication protocols [12], thereby leading to a vast attack surface. As a result, heterogeneity is considered as being one of the largest issues impeding IoT security today.

This paper introduces EclipseIoT, a novel secure and adaptive hub which enhances the overall security of the IoT ecosystem. More specifically, EclipseIoT is a secure hub which addresses heterogeneity by supporting both commercial and embedded IoT devices, regardless of their vendor. The hub's architecture consists of: (1) a gateway which allows users to access their smart devices over a secure channel, eliminating data leakage, (2) a policy server which provides policy-based access control, eliminating unauthorised access on the network, (3) a network configuration that limits the exposure of local IoT devices through *cloaking*, which reduces the risk of identifying or attacking the smart devices, and (4) a *canary-function* based capability, which allows the behaviours of attacks to be monitored. The evaluation of this system demonstrates that a large percentage of the attacks which affect the traditional IoT ecosystem are mitigated. The paper is organised as follows: Section 2 focuses on the related work, Section 3 describes the overall architecture of the hub and its individual components, Section 4 details the implementation of EclipseIoT, Section 6 presents the system evaluation, and Section 7 concludes the paper and discusses our proposals of future work.

2. Related Work

Several studies which concern IoT security focus on solving specific aspects such as authentication, confidentiality, integrity, and access control. For example, Zhao and Ling [13] improve authentication by using a custom encapsulation mechanism which combines cross-platform communication with encryption. Moreover, Kothmayr et al. [14] introduce IoT's first fully implemented two-way authentication security scheme, which is based on Rivest-Shamir-Adleman (RSA) and designed for Internet Protocol version 6 (IPv6) over Low power Wireless Personal Area Networks

(6LoWPANs). Liu et al. [15] and Pranata et al. [16] address confidentiality and integrity by presenting a framework based on Public Key Infrastructure (PKI). Finally, Ye et al. [17] and Ma et al. [18] develop lightweight access control systems for the IoT ecosystem.

Others have studied IoT hubs, which are currently among the most popular IoT management models [19]. The fundamental idea of IoT hubs is to allow sensors with low computational power to focus solely on the task they are designed to do (e.g. measuring the temperature), without having to consider more complex functions (e.g. network connectivity, data processing, etc.) [20]. These functions are handled by the core of the hub, the gateway, which has significantly more computational power and storage in comparison to the sensors [19]. In addition, IoT hubs have the capability of connecting a wide range of diverse smart sensors (e.g. thermostats, smart locks, light sensors) by using wireless technologies. However, such implementations have their limitations such as limited inter-operation capabilities between devices from different vendors and protocol heterogeneity. In order to address the requirements needed by the communication interface, programming abstraction, and community-driven device support, Mozzami et al. [21] implemented the XML based framework, SPOT. SPOT uses device drivers to support additional devices on the framework. However, this technique can be time-consuming and not user friendly. Their framework includes OAuth as a security mechanism, yet, their framework's overall performance is not evaluated.

Saxena et al. [22] propose implementing an IoT gateway which is expected to operate over 5G networks, with the main focus being on resolving the challenge of resource constrained wireless networks. However, this approach does not address the core security concerns, and similar to Mozzami et al. [21], does not evaluate its feasibility. Gloria et al. [23] suggest an IoT gateway that addresses IoT heterogeneity by supporting all the available wired and wireless communication protocols. However, this approach focuses only on low-level embedded devices and does not support commercial smart home devices. Additionally, this solution does not focus on security, and the authentication mechanism that is employed is primarily for platform configuration purposes. To tackle heterogeneity and to control the large data volume generated by IoT, Razafimandimby et al. [24] propose a Bayesian Inference Approach (BIA). This model is heavily dependent on sub-nets of IoT devices and smart gateways, which use probabilistic techniques to avoid sending *useless* data to the network. Although this approach is successful in reducing resource consumption, it does not address the security of the IoT ecosystem. Alsheri & Sandhu [25] propose an architecture where

virtual objects use publish and subscribe (pubsub) topics to deliver services to users through a security policy. Although this approach is successful in eliminating scalability and heterogeneity, it does not provide any security mechanisms to the IoT ecosystem.

Commercial implementations such as *Samsung Smart Things*, *Apple HomeKit* and *Google Smart Home* provide similar solutions to Mozzami et al. [21], and are capable of supporting function limited devices from various vendors. However, these frameworks are not able to inter-operate with each other. Other technologies from vendors such as IFTTT, Nest Thermostat, ALlJoyn, and HomeKit [21, 26, 27, 28, 29, 30] do not provide third-party extensibility or security mechanisms. As a result, they cannot provide a unified interface, which further demonstrates that heterogeneity within IoT products and services is a current problem. Guoqiang et al. [31] propose a centralised framework which employs a gateway which allows different communication protocols to be plugged in to support different networks. Nevertheless, this approach requires advanced configuration techniques and changes in the physical hardware, which can be costly and is not easily extensible. Finally, the framework does not include a security policy or other mechanisms to improve the security of IoT devices.

It is therefore evident that previous propositions, which aim to tackle the heterogeneity in IoT, do not specialise in improving the security of the ecosystem. Instead, the focus is on handling the big data that is generated by smart devices, attempting to overcome constraints of their computational power, and their scalability. While security is a key concern within these existing approaches, the supported security requirements are limited and are partially addressed. To summarise these approaches, Table 1 describes existing frameworks, along with the security features they provide. To the best of our knowledge, EclipeIoT's built-in security mechanisms are novel, tackling heterogeneity and enhancing the security of an IoT system.

Implementations	Authorisation	Authentication	Encryption	Access Policy	Early-Detection Security	Multi-Vendor	Mult-Type Devices	Local-Network API Support	Cloud-based API Support	Third-party Extensibility	Unified Interface	Uniform Request Format
	Features											
SPOT		✓				✓	✓	✓	✓	✓	✓	✓
Saxena *et al.*			✓			✓	✓	✓	✓		✓	
Gloria *et al.*		✓				✓	✓	✓				✓
BIA								✓	✓			
Alshehri and Sandhu				✓		✓	✓				✓	✓
Guoqiang						✓	✓				✓	✓
EclipseIoT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Existing implementations often neglect aspects of security.

3. EclipseIoT Hub

3.1. Desired Properties

The primary function of the proposed framework, EclipseIoT, is to address the heterogeneity in an IoT ecosystem. In doing so, the hub must be able to support both embedded and commercial devices regardless of their vendor. Secondly, the framework must provide various mechanisms to enhance the overall security of the IoT network. Such mechanisms include authentication, transport encryption, limiting the exposure of IoT devices on the network, monitoring attack behaviours, and provide access control.

In providing the above functions, the proposed system aims to fulfil several properties such as:

1. **Adaptiveness:** The hub should support various diverse devices.
2. **Authentication:** An adversary should not be able to obtain unauthorised access to the IoT network.

3. **Confidentiality:** The communications between the hub and the IoT devices should be encrypted.
4. **Transparency:** The hub should camouflage the IoT devices, in order to increase the difficulty for an attacker to locate them.
5. **Monitoring:** The hub should provide a mechanism to monitor attack behaviours.
6. **Access Control:** The hub should provide an access control mechanism to further enhance the security of the system.

3.2. Architectural Components

In order to achieve the above functions the proposed IoT hub consists of two main components, a gateway and a policy server (Figure 1). The gateway is the core of the system and is capable of communicating with each device, whilst also allowing users to access their devices over a secure communications channel. Simultaneously, the policy server maintains accountability of such access.

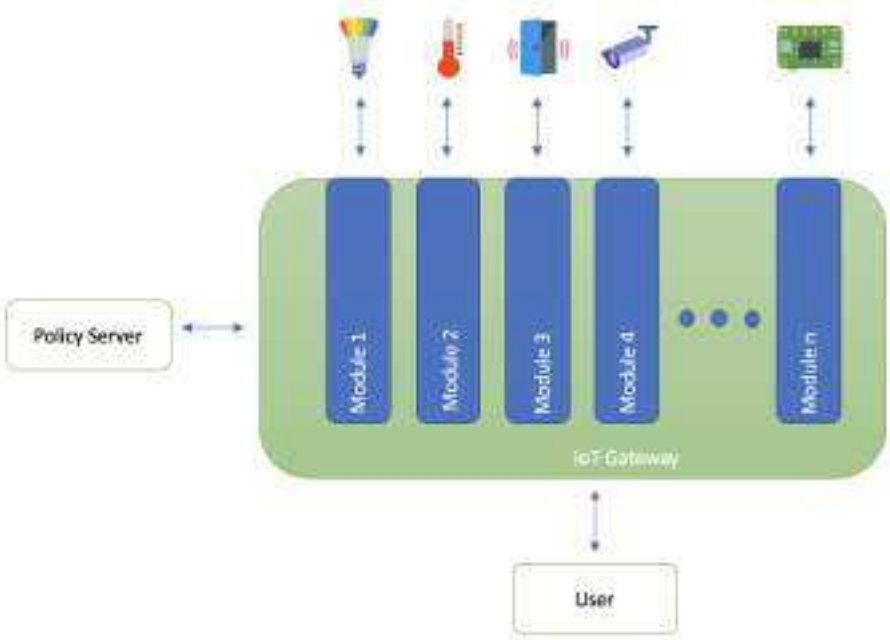

Figure 1: EclipseIoT infrastructure consisting of a gateway and a policy server

Once a user requires to operate a device which is connected to the network, their request is forwarded to the gateway. Prior to this request being fulfilled, it is redirected by the gateway to the

policy server. The policy server queries the policy database, checking whether the request satisfies predefined attributes associated with the user making the request. Depending on the result, the policy server grants or rejects the request.

4. Implementation

In this section, the implementation of EclipseIoT hub prototype and its application within a smart home testbed are discussed. In EclipseIoT, the gateway controls all the connected IoT devices using PubNub [32] and add-on modules which are written in Python. PubNub is an API which employs a Data Stream Network (DSN) and allows users to connect to it, make requests, and interact with the connected devices. The hub/gateway supports the ability to interface between multiple devices by allowing third-party developers to implement add-on modules to control them. A repository system can also be adopted, allowing users to download modules for a device they wish to use in their IoT ecosystem. The policy server verifies the user's requests.

EclipseIoT provides a middleware solution which uses a Publish-Subscribe (Pub-Sub) architecture with security support. More specifically, a publisher (i.e. any source of data) pushes messages out to interested subscribers (i.e. receivers of data) via live-feed data streams known as channels (or topics). All subscribers of a specific publisher channel are immediately notified when new messages have been published on that channel, and the message data (or payload) is received together with the notification [33]. This technique allows users who interact with IoT devices to authenticate with the gateway using a 3-step process, which ultimately creates a secure channel. The channel is analogous to a topic, as described above, in which the gateway and the user both contain the permission to publish and subscribe. Requests by the users and responses from the gateway are securely passed through PubNub's DSN.

Although the gateway and the security policy provide tighter security controls and a more unified platform to interface with various heterogeneous devices, to further enhance the security of the framework, there is a need to implement a mechanism to obscure the IoT devices connected to the network. To achieve this, and to ensure that requests to the IoT devices are only made through the gateway, a specific network configuration was implemented. During this configuration, the smart devices were isolated within a sub-network, whilst the gateway remained accessible through PubNub client application over the Cloud. This has a key role in improving the security of the system, as IoT devices can be considered to be the weakest link for breaking into a secure infrastructure.

Therefore hiding/isolating such devices makes it far more challenging for an attacker to locate a device and discover its vulnerabilities. The key component that made this network configuration possible is the gateway, as without it, devices that require the user to be on the same network to operate them would lose functionality. Table 2, summarises the key components of EclipseIoT, their functionalities, and the security aspects they enhance.

Component	**Functionality**	**Advantage**
Gateway	Authentication	Enhanced Access Control
	User Requests	State change, securely, and uniformely
	Module creation remotely	Post-deployment configuration
	IoT tailored canary files	Early unauthorised access detection
Policy Server	Accepts/Rejects requests	Enhanced request control mechanism
	Add/Edit/Delete a policy remotely	Post-deployment configuration
	Create canary functions remotely	Post-deployment configuration
	Manage canary triggers	Alerting system administrators, blacklisting, etc.
Sub-Network	Camouflages the devices and framework	Enhances the security
	Device accessibility	Devices can be accessed from anywhere

Table 2: EclipseIoT components and their functionalities

4.1. Service Provider - Pubnub

In EclipseIoT, communication links between the gateway and the user and the gateway and the policy server take place with the assistance of a cloud service provider. Currently, there is a range of cloud services available which allow users or IoT devices to interact with each other. Such service providers include: Dweet.io [34], KURA [35], Zetta API [36], and Pubnub [32].

Dweet.io does not give any guarantees of their service to be suitable for enterprise-level security such as adhering to security protocols or encryption. As a result there may also be privacy concerns using this service, as they state the previous five messages sent from IoT devices through their service are held for 24 hours, and all messages are public unless a "lock" feature is purchased. Due to these issues Dweet.io was dismissed as a service that we would utilise for this work. Furthermore, KURA is described to be usable for IoT gateways using the Raspberry Pi. However, the software has been open-sourced, and as a result, the support is not very consistent. Additionally, there were

also some compatibility issues associated with it. Finally, there were few security concerns that could not be addressed with it. Although KURA provides Transport Layer Security (TLS), it does not allow provisioning security through the cloud or configure security post-deployment. Thus, due to compatibility and configurable security issues, further work with KURA was not continued. Ultimately, Zetta is another open-source software that allows IoT servers to be connected through the cloud. Nevertheless, its focus is big data oriented rather than security, consequently it can not be used to built EclipseIoT.

After carefully considering the advantages and disadvantages of each service, Pubnub seemed to be the most suitable upon which EclipseIoT could be built. That is because it provides options to configure security, to connect various IoT devices, to create multiple IoT gateways, and finally it supplies well-documented references for a range of programming languages and operating systems. More specifically, Pubnub is an Infrastructure as a Service (IaaS) DNS, that allows users to control IoT devices, by providing monitoring and security provisioning capabilities. Using Pubnub, various heterogeneous IoT devices, sensor networks, hubs, and other electronic devices can be connected together. Via this service specific actions can be triggered to any of the connected devices, whilst additional monitoring and processing can occur in their in-going and out-going data securely [32]. Pubnub is an infrastructure that allows API communication between such devices in a network of any size. For the purpose of this work, Pubnub's free tier was employed. It provides service for up to 100 devices and 1 million messages per month, as well as basic support, which within this work has been instrumental.

4.2. Heterogeneity & Module-based Adaptation

EclipseIoT's primary function for handling device heterogeneity is the inclusion of add-on modules within the gateway. These represent the interfaces of each smart device connected on the network and are responsible for various operations. The add-on modules are implemented in Python and support both commercial and embedded devices. The only prerequisite for these to work, is that the devices need to have an API available. For every device connected on the network, each module must implement a `get_mac` function in order for the gateway to identify the MAC address for each one. Furthermore, these modules perform API calls using Python *requests* module to execute various functions. All the results from these requests are received in JSON format and parsed with the Python `json` module.

Few of the commercial devices that were used (e.g Philips Hue Lamps), have an API available, and therefore we were able to develop an add-on module which was responsible for functions such us: adjust brightness of the lamp, show available lamps, switch the lamp on and off. The TP-Link NC200 camera did not have an official API and we used a third-party developed API for a similar model. As a result, due to compatibility issues we only managed to implement an add-on module that takes a snapshot, uploads the image on a website, and forwards the Uniform Resource Locator (URL) to the user. Finally, the Samsung Smart Things and LG smart TV, provide a very detailed API. However, they require different programming languages for the communication. To address this issue, we implemented Python wrappers. This allowed us to implement add-on modules for all different types of devices/sensors that are supported by the Samsung Smart Things hub such and the smart TV. Specifically, the sensors available for the Samsung Smart Things were a motion sensor, two switch devices, a smart plug, and smart bulbs. To integrate the Python wrapper, an add-on module smart_things was written with the coherent functions: list_types, list_devices, find_of_type, toggle_switch, and device_state. For the smart TV an add-on module featuring the following functions was also implemented: `get_volume`, `set_volume`, `volume_up`, `volume_down`, `open_url`, `launch_app`, `launch_app_with_params`, `get_apps`, `current_app`, `close_app`, `get_services`, `get_inputs`, `set_input`, `switch_3d_on`, `switch_3d_off`, `power_on`, and `power_off`.

As an example of an embedded device in our testbed, we used a Raspberry Pi 3 that controlled few LED lights. In order for the gateway to be able to communicate with it, we installed and run PubNub on it as an end-device. This made the Raspberry Pi capable of receiving instructions from the EclipseIoT gateway. These instructions can be sent using the `embedded_devices` PubNub channel, to control hardware such as the LEDs. The EclipseIoT Gateway is subscribed to the `embedded_devices` channel and the `controlpi.py` module allows LEDs to be turned on or off, blink, as well as be able to send messages in Morse code by flashing the LEDs.

The above demonstrate that add-on modules, for a range of commercial and embedded devices, can be written and installed in EclipseIoT, making possible for it to communicate with heterogeneous devices. Finally, EclipseIoT provides users secure remote access, even on devices or APIs normally supporting only local network interaction, such as the wrappers mentioned above. However, these wrappers were enhanced through the use of our framework, by providing communication through PubNub, allowing users to make requests from anywhere in the world, through the internet.

5. Security Mechanisms

5.1. Incorporating IoT Devices in a Sub-Network

In a typical home, traditional and smart devices gain Internet access by connecting to the same Access Point. Figure 2 illustrates an example of a conventional network topology of such an ecosystem. Specifically, it consists of two Access Points (routers); one central router provided by the Internet Service Provider (ISP) and a second third-party ASUS router. The majority of the IoT devices are connected to the access point by a power line adaptor, via Wi-Fi, or Ethernet. Vendor specific devices/sensors, such as the Philips Hue lamps and Samsung Smart Things motion sensor, are connected to their control hubs via Zigbee or Bluetooth Low Energy (BLE). However, although this is the most popular network configuration, it is vulnerable to various wireless attacks such as passive sniffing, Man-In-The-Middle attacks, ARP/MAC spoofing, etc. Given that all the smart devices are exposed to the network, the security risks are significantly increased.

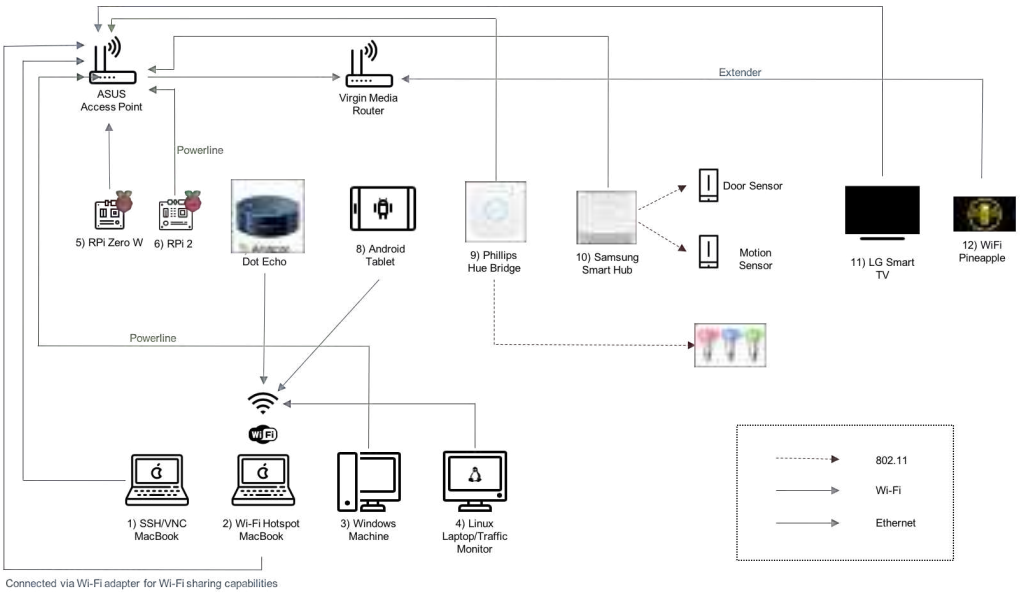

Figure 2: Typical IoT network which illustrates multiple means of connection (i.e. power-line, Ethernet, Wi-Fi and Wi-Fi hotspots).

To address the above issue, Figure 3 illustrates a proposed network configuration, which includes several additional components which sit within a sub-network to enhance IoT security and interoperability (i.e. a gateway, a policy server, and a policy database). More specifically, a second Access

Point (router) was used, to create a sub-network to which IoT devices, irrespective of manufacturer, can connect and communicate seamlessly and securely via the novel EclipseIoT implementation. Within the sub-network, the gateway of EclipseIoT is now the core of the network and the only externally visible device to the world, via PubNub service provider. Moreover, the gateway is also the main point of communication of rest of the IoT devices that sit within the sub-network. The sub-network configuration provides additional security, as users who join the main/local network (192.168.0.x) can no longer view the connected smart devices connected as they are camouflaged. This is due to the "hidden" setting of the sub-network, which makes it invisible. Furthermore, devices on the sub-network, can no longer connect or access other devices on the main network, due to Internet Protocol Access Listings (IP ACLs) configurations on the sub-network's router. From a security perspective, this configuration reduces the risk of attacks that are targeted towards vulnerable devices (e.g. default password attacks), as well as aligning IoT networks with typical home/corporate networks, mitigating attacks targeted towards IoT devices. Finally, it protects the main network in case that vulnerable IoT devices get infected. The use of Quality of Service (QoS), or route-maps, can also be employed on the sub-network's router to further enhance the security by controlling the flow of network traffic, allowing network administrators to identify threats and close points of entries.

The inclusion of a gateway within the IoT network enhances heterogeneity and interoperability. One may assume that the addition of a new sub-network may cause the functionalities of and interactions with the included IoT devices to be lost. For instance, Philips Hue lamps must be on the same sub-network as their interacting devices. Changing lighting settings with a smartphone is therefore not possible from the main home/corporate network. However, the inclusion of a gateway resolves this issue as it receives its Internet connectivity through Pubnub's cloud platform. Consequently, a user can access the gateway via a secure channel from the main home/corporate network and can control the smart devices from anywhere via the Internet.

At this point, it is worth discussing the extensiblity of EclipseIoT's network configuration. The framework is flexible, allowing new devices to be added when required. However, such devices must be compatible with the EclipseIoT server and must have an available API so that add-on modules can be implemented to support its inclusion into the system. If the device's firmware changes, it still must be compatible with the add-on modules. If this dependency fails, the device will become inaccessible and administrators would need to perform inconvenient periodic tests to ensure that the

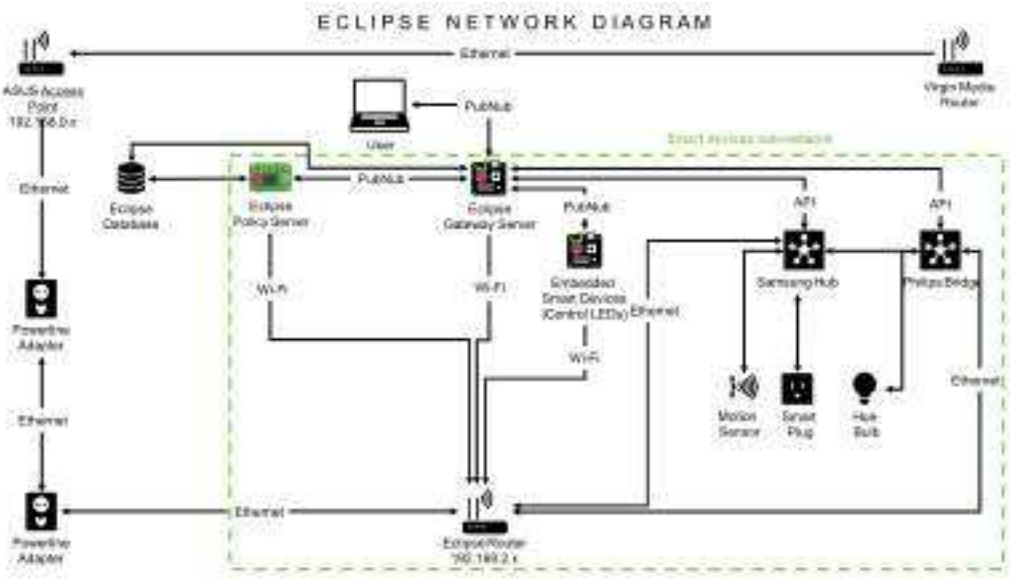

Figure 3: The new network configuration contains a specialised sub-network used for cloaking/isolating the smart devices. These communicate with the outer world via EclipseIoT gateway.

device's firmware is still compatible. Additionally, due to the centralised and currently synchronous architecture of EclipseIoT, more devices on the sub-network means more requests are being made to the gateway. As a result, the overhead may increase resulting in possible blackouts.

5.2. Authentication and Security Protocols

Prior to sending instructions to EclipseIoT's gateway, which will allow users to get or set the state of a smart device, users are required to complete a three-step authentication process (see Figure 4). Although PubNub provides the main functionality for this process, the proposed system has been further configured to also employ AES256 algorithm, in order to achieve confidentiality. More specifically:

1. Firstly, the user is required to join the `gateway_auth` channel by simply subscribing to it. Once the client is subscribed to this channel, the gateway is able to detect the presence of the user. Afterwards, the gateway creates a channel where the channel's name matches the user's Unique User Identification (UUID), henceforth called UUID channel.
2. Once the UUID channel is created, the gateway computes the hash of the user's UUID number and forwards it back to the user over the `gateway_auth` channel, which notifies them that

their channel has been established. When the user receives the hash from the gateway, unless it is already computed, the hash of their own UUID must be created and compared with that which is received.

3. If the two hashes match, the user is able to join this specific UUID channel, establishing a one-to-one communication with the gateway.
4. At this point, the gateway must ensure that only one user is on the channel, and that the correct user is on the correct channel by checking the user's UUID. If these two conditions are met, the gateway will send an authentication key to the client over the UUID channel. This particular configuration also allows for the AES256 key to be sent over the same channel, allowing the user to send encrypted requests.
5. At this point it is worth noticing, that when the secure channel is created using PubNub's Access Manager (PAM), which also uses AES, it is locked and only this randomly generated authentication key can be used to gain access to it.
6. Lastly, the secured communication is formed by using a secure/locked channel, which utilizes both the authentication and AES256 encryption. The TLS protocol is used throughout the duration of this process.

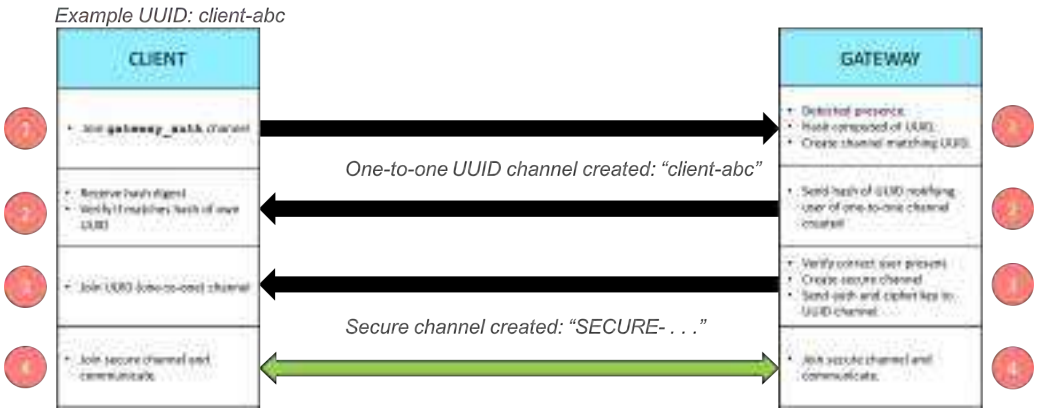

Figure 4: The four phases included within the three-step authentication process required to establish a secure communication channel

5.3. Security Policy Server

The security policy server operates separately to the gateway. Each request, sent by the users, calling to interact with the smart devices within the sub-network is forwarded to the IoT gateway. Before being fulfilled, the requests are sent to the policy server. The main role of the policy server is to ensure that each request complies with the policy that has been defined on the server. If the request counters these policies, it is rejected. The `gateway_policy.py` and `policy_database.py` fulfills this task together. The following policies are included in the current implementation of the policy server:

- The time of the request is within the acceptable access time-frame for the particular module.
- There is a time policy defined for the module/function the user wishes to access.
- The user must not be blacklisted from: (a) all modules/devices; (b) the module that implements support for the requested device; and (c) for the specific function of the module belonging to the device of interest.
- The user must not be trying to access a *canary function*.
- The user has not been rejected access three or more times on the same day [37].
- The user's latest access was not rejected under a minute prior to the current access request [37].
- An incorrect user, who has been granted authorization for the channel and who is not requesting access, is blacklisted. [37].

The current implementation of the policy server allows for a versatile remote configuration. This enables administrators to remotely run `policy_database.py` functions such as `set_policy` and `modify_policy`, which simplify remote policy updates. The `gateway_policy.py` listens to such messages on the `admin_channel`, subsequently executing the functions in `policy_database.py`. This is one of the design principles used in this work to ensure that there is some level of independence between the implementation of the EclipseIoT server, and the definition/specification of a policy that such a server uses. Another important aspect of `gateway_policy.py` is its ability to receive messages from the `gateway_receiver.py`, which contain the request made by the user.

This request is verified by the `gateway_policy.py`, which queries the database for various security attributes, e.g. the time policy, blacklists, etc. Once the result is determined, it is presented to the `gateway_receiver.py` which either executes the IoT function that the user had requested, if it fulfills the policies, alternatively it informs the user that their request was rejected. The `gateway_policy.py` logs all access attempts.

5.4. IoT Canary Functions

In order to entice a potential attacker and to enhance the security of EclipseIoT, we designed and implemented canary functions which are tailored specifically for IoT. These functions are based on the canary files used within traditional IT systems. A canary file is a forged file, which is typically placed amongst genuine ones in order to support the early detection of unauthorised data access, copying, or modification – and work in a similar way to a server side honeypot. Its name originates from the use of canaries as sentinel species within coal mining environments to warn workers of the build up of gasses such as carbon monoxide underground [38]. In this context, if an unauthorised party attempts to interact or gain access to these functions, an action is triggered. In the proposed implementation, bogus add-on modules and functions that represent sensitive operations of IoT devices, are being created and deployed. These are displayed to users as regular module functions. For example, a module named `SmartLock.py` is assumed to operate a smart lock, and it contains a function called `get_pin()`. In order for the user/developer to decide which functions should be implemented as canaries, they initially need to identify and list all the operations for each IoT device. Given that canary functions need to represent sensitive/important functions in order to lure attackers, a further evaluation of the operations for each device according to their severity is necessary. With this in mind, developers have the choice to only implement the most severe operations as canaries, or as in the proposed implementation, various grades that correspond to different actions can also be designed.

More specifically, the severity of the canary function is classified according to one of the following three categories: A: most severe, B: average severity, and c: the least severe. A function classified as grade A would prevent unauthorized access by causing the system to shut down and notify an administrator via email. A grade B canary would notify the administrator via email and blacklist the user from further access. Finally, a grade C function would only notify an administrator via email and no further action would be taken. As previously described, once a user sends a request to

access a function, the gateway forwards it to the policy server. In cases where users have requested to access functions which are enlisted as being bogus, the policy server will query the database to inspect the severity of invoking a canary function and proposes a possible action.

All the attempts for accessing canary functions are being logged into a text file. This is beneficial because these text files can be further studied, in order to understand the intentions and interests of the attackers. As a result defence mechanisms and countermeasures can be selected with more accuracy and efficiency. Furthermore, Machine Learning (ML) algorithms can also be employed to help understand and analyse intelligently attacker's behaviour, by recognising patters. ML algorithms could even assist in predicting attacks. Finally, one of the novel features of this implementation is the employment of visibility controls. These controls define which modules containing canary functions are displayed to which users. For example, the module `NestThermostat.py` will only be displayed by the gateway to users that intend to access it. This is specified by the database table's UUID attribute.

5.5. Home-based Testbed

To examine the frameworks' feasibility, its performance is being evaluated by applying it to a smart home IoT testbed, which consisted of a range of commercially relevant and representative IoT hardware. Such devices included a TP-Link NC200 IP camera, the Samsung SmartHub with a connected motion sensor, a TP-Link Smart Plug, and the Philips Hue starter kit. The gateway and policy server were installed onto a Raspberry Pi 3 and a Raspberry Pi Zero Wireless system respectively. An additional Raspberry Pi 3 was used to emulate a user device which controlled LED lights. Figure 5 displays the smart home testbed setup used for the experiments. Devices in orange are commercial controllers, those labelled in green are end-devices which are controlled by sending EclipseIoT gateway commands, which are verified by EclipseIoT's policy server which subsequently sends commands to the controllers.

6. Evaluating the Security of EclipseIoT

In order to evaluate the security of EclipseIoT, several attacks were performed on both the traditional IoT network and the network on which EclipseIoT hub had been deployed. The main focus of these attacks was on vulnerabilities associated with traditional IoT networks such as network sniffing, MITM, DoS, Spoofing, etc. [39]. In an attempt to use a non-biased approach, a third party

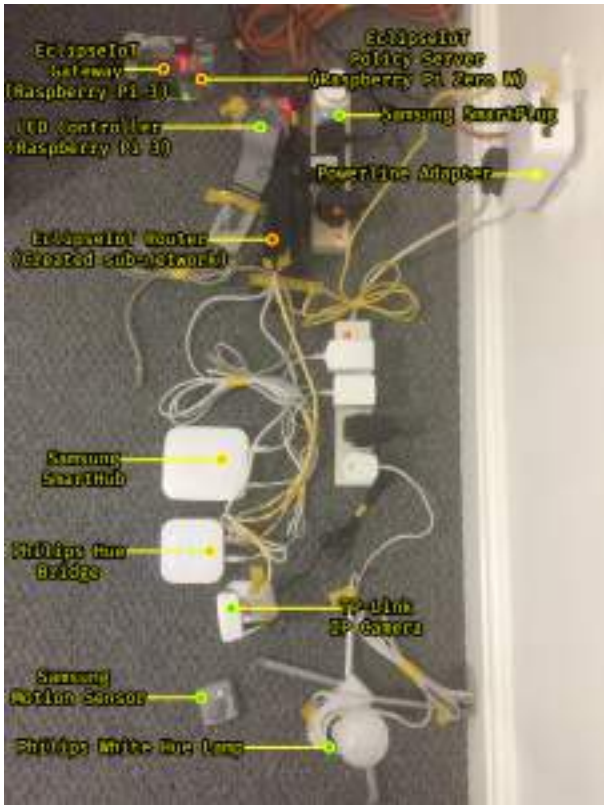

Figure 5: Applying EclipseIoT to a smart home IoT testbed

certified penetration tester was adjudicated. For each attack, it was assumed that an attacker had gained access to the main network.

Firstly, in order to discover all the devices and possible sub-networks connected to the main network, including their MAC addresses and BSSIDs, the network was scanned using the *airodump-ng* tool [40]. In the traditional IoT setup, three access points/networks were identified, including the one we were already connected to. By focusing on this access point, all connected IoT devices/clients became immediately visible. On the other hand, once EclipseIoT was deployed, the tool identified four access points/network, two of which were hidden. By focusing on the main network's access point, only the connected traditional IT devices, such as laptops and desktops, were revealed. The challenge at this stage was to iterate through each network to discover which contained any smart devices. By limiting the range of the EclipseIoT sub-network, it would become more difficult to detect it. Moreover, by using a combination of the *airodump-ng* tool and information about

manufacturer flags, it was also possible to expose the sub-network's router SSID. In order to test the reachability of this router, an *Internet Control Message Protocol (ICMP)* echo. However, there was no response from the main router as it was disabled in its configuration. Finally, following the success of identifying the MAC address of the sub-network's router, the next step was to attempt to capture the WPA handshake in order to gain access to the sub-network. Subsequently, 4 de-authentication packets were sent to the router. Once the clients were disconnected and reconnected, the handshake was captured and the hash of the password was taken. However, at this point brute-forcing the passwords was unsuccessful.

To gain further information about the network topology and the devices that are connected, *Zenmap* and *Nmap* [41] were used. In the traditional IoT network, it was possible to reveal the topology and information about all the devices connected on the network, such as IP addresses, names, and open ports. It was noted that few of the IoT devices had between three and six ports open. On the contrary, when EclipseIoT was deployed, it was only possible to gain information about the traditional IT devices that were connected on the main network. These demonstrated to also have SSH and HTTP ports open. However, it was not possible to view and gather additional information about other smart devices on the network. In order to identify if any of the Access Points employed Wi-Fi Protected Setup (WPS), for both networks, the local network was scanned by using *wash* command. This revealed that EclipseIoT's sub-network's router did indeed employ this mechanism. In the future, we would have to ensure that the WPS is disabled.

As all devices on the traditional IoT network were exposed and had previously successfully gained the IP addresses, it was possible to perform Denial of Service (DoS) attacks to any smart device by using the *deauth*[42], *Ping flood*, and *hping3* [43] tools. On the contrary, when EclipseIoT was deployed, IoT devices became hidden within the sub-network, increasing the difficulty in identifying them and performing DoS attacks upon them. Therefore, in an attempt to detect if any clients were connected to the sub-network's router that was successfully identified, *airtplay*[44] tool was used. In order to perform this attack, it was necessary to possess the router's MAC address and password. Although it was possible to gain the MAC address, for the purposes of this attack, the pentester was also provided with the password as they were unsuccessful in brute-forcing it. Finally, the pentester successfully revealed the client/IoT devices connected on the sub-network's router and managed to spoof the MAC addresses of the IoT devices.

DoS attacks often leverage ARP spoofing to link multiple IP addresses with a single target's

MAC address. As a result, traffic that is intended for many different IP addresses will be redirected to the target's MAC address, overloading the target with traffic [45]. Given that it was possible to successfully gain information regarding all of the devices that were connected to the traditional network, ARP spoofing was successfully performed in multiple smart devices, using *arpspoof* [45] tool. However, when EclipseIoT was deployed, it was not possible to gain enough information regarding the connected devices, leading to an unsuccessful attack.

Subsequently,there was an attempt to intercept sensitive data by passively sniffing the network. In the traditional IoT network, all IoT devices apart from one, employed the TLS protocol and therefore intercepting any traffic in plaintext was not possible. When concerning the one device that did not employ TLS, it was possible to successfully intercept the credentials necessary to log in onto the devices's web interface. At this point, Man-In-The-Middle attack was also performed, in an attempt to bypass the encryption that TLS provided. As a result, sensitive data in relation to the smart devices such as usernames, passwords, and location coordinates, was successfully intercepted. When EclipseIoT was deployed, another attempt of passively sniff the traffic on the main network, specifically in the communication channel between the sub-network's router and the user where the user/gateway requests are transferred, was made. However, as EclipseIoT employs the TLS protocol and AES algorithm, it was not possible to intercept any traffic in plaintext. Following this, an Evils Twin attack, which is a type of MITM, was attempted in order to strip the TLS. This also demonstrated to be unsuccessful.

Conclusively, the above experiments demonstrate that EclipseIoT significantly enhances the overall security of the IoT ecosystem, as it mitigated against most of the attacks that are traditionally deployed on a conventional IoT network set up. Due to the sub-network's configuration, an attacker was not able to identify any smart devices connected on the main network, spoof their MAC addresses, or sniff their data. A summary of these attacks as well as their results in both network typologies can be found in Table 3.

6.1. Performance-driven Configuration

To measure the performance of EclipseIoT the execution time of the round-trip of the user's request to the gateway and back, was measured. As EclipseIoT has two variants of implementations, one that employs only the TLS protocol and another that employs the TLS in combination with AES algorithm, we assessed and compared the performance on both cases. In each case, the implemented

Attack	Attack Method	Target	IoT	EclipseIoT
Device Detection	airodump-ng/Nmap	Network Identification	✓	✗
MAC Spoofing	airodump-ng	System Blackout	✓	✗
ARP Spoofing	arpspoof	System Blackout	✓	✗
Passive Sniffing	Wireshark	Data Leakage	✓	✗
MITM	Evils Twin	Data Leakage	✓	✗
DoS	deauth, ping flood, hping3	System Blackout	✓	✗

Table 3: Attacks used to evaluate the security of both the traditional network and the network once EclipseIoT is deployed. The check and x markers denote than the attack was successful and unsuccessful respectively.

add-on modules contain several functions that serve different operations. Therefore, as each one has it is own execution time, we tested them separately. Each function was executed 100 times and the average time was calculated in seconds (s). For instance, Figure 6 illustrates the execution time of the `lg_tv.py` module which operates an LG smart TV and contains the largest amount of functions in the network. Specifically, *get_volume*: displays the current volume, *set_volume*: changes the volume, *volume_up*: increases the volume, *volume_down*: decreases the volume, *open_url*: opens the TV's browser launching the requested url, *launch_app*: launches an application (e.g. Netflix), *get_apps*: displays all the available apps on the TV, *current_app*: displays the current app that is open, *close_app*: closes an app, *get_services*: displays vendor related services (e.g. tv guide), *get_inputs*: show available inputs (e.g. HDMI1, HDM2, etc.), *set_input*: switch to a specific input, *power_on*: turn the TV on, *switch_3d_on*: turns on the 3D, and finally the *switch_3d_off*: turns the 3D off.

Convincingly, according to the above results, we demonstrate that EclipseIoT's performance varies according to the configurations that the user has selected. Specifically, the delay is greater when more security mechanisms are employed, in this case specifically, when TLS is used in combination with AES. However, it is up to the user to decide which of these mechanisms should be supported given the additional overhead. Nevertheless, the performance of Eclipse IoT could be improved. Currently when a request is being processed by the gateway, few other functions can not operate at the same time and therefore there is a delay. Additionally, the performance of the system may also get affected in a larger network. Both of this issues though could be addressed by

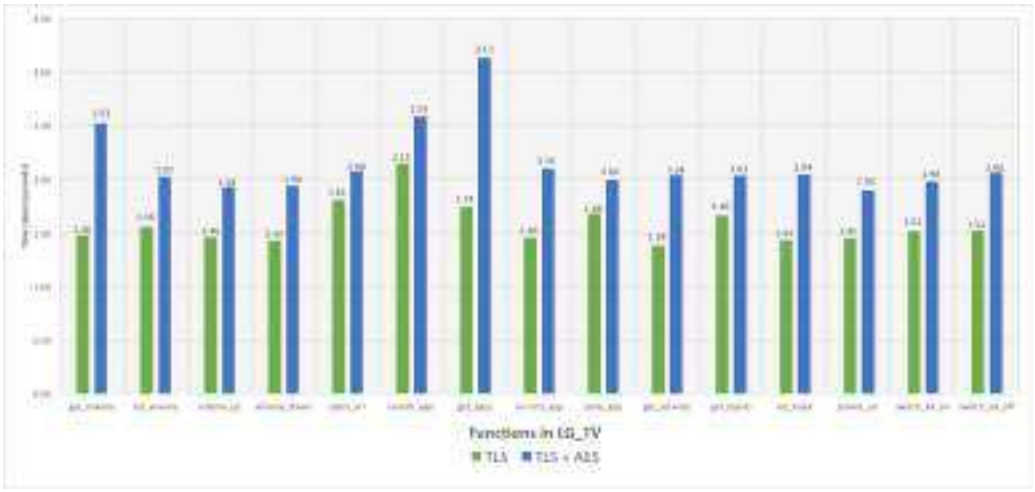

Figure 6: Execution times of the functions contained within the lg_tv.py, that controls the LG smart TV

making the framework operate asynchronously.

7. Conclusion

This paper introduces EclipseIoT, a hub which aims to address IoT heterogeneity, as well as enhancing the overall security of a smart environment. The main components of EclipseIoT are both a gateway and a policy sever. The gateway is capable of communicating with each device whilst also allowing users to access their devices over a secure communications channel, addressing the heterogeneity of the IoT ecosystem. Simultaneously, the policy server maintains accountability of such access. Further mechanisms such as authentication, AES256 algorithm, sub-network configuration, and canary functions also enhance the overall security.

In order to evaluate the security of the proposed hub, it was implemented and further incorporated within a home-based testbed. This included commercially available devices, to which we applied a penetration testing methodology consisting of a selection of various attacks. The results from such attacks demonstrated that EclipseIoT significantly improves the security of the heterogeneous IoT ecosystem, as it was able to mitigate against most of the attacks which affect conventional IoT networks. The proposed framework is available for download at: `https://goo.gl/bap97h`. However, EclipseIoT faces its limitations. Firstly, it relies on third-party providers, such as PubNub, to support some actions within the framework. If such third-parties halted their services, the

actions on our framework would not be able to perform. Another limitation is the fact that the communication channels between the gateway and the user and the gateway and the policy server, are based on TLS across PubNub. The security of the system was enhanced by implementing the AES256 algorithm in addition to the TLS protocol. However, as these communications are passed over PubNub, they must be decrypted to be able to forward messages to the correct destination. When using AES alongside TLS, PubNub is not able to perform this action. Lastly, although APIs provide an accessible and user-friendly interface to access, add, and control the smart devices, they are often subjected to having limits for the number of requests that they can receive.

Given EclipseIoT's initial positive performance, it is possible to scale up the framework and subsequently improve it's performance by integrating and monitoring other architectural aspects. For example, investigating whether installing a local broker (e.g. Mosquitto (mqtt)) to the gateway may increase the performance of EclipseIoT, in comparison to the third-party providers (e.g. PubNub), which are currently employed. Furthermore, expanding the threat model so that IoT devices are protected from one another and not just from adversaries on the other side of the network, would also enhance its security. Employing micro-segmentation or 12 fire-walling so that devices can only communicate with the gateway should help achieve this.

References

[1] K. Ashton, That internet of things thing, RFiD Journal 22 (7) (2009) 97–114.

[2] Gartner says 6.4 billion connected "things" will be in use in 2016, up 30 percent from 2015, `https://gartner.com/newsroom/id/3165317`, (Accessed on 11/18/2017).

[3] T. Xu, J. B. Wendt, M. Potkonjak, Security of iot systems: Design challenges and opportunities, in: Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design, IEEE Press, 2014, pp. 417–423.

[4] Internet of things research study, hewlett packard enterprise, `http://files.asset.microfocus.com/4aa5-4759/en/4aa5-4759.pdf`, (Accessed on 12/29/2017).

[5] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): A vision, architectural elements, and future directions, Future generation computer systems 29 (7) (2013) 1645–1660.

[6] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, S. Shieh, Iot security: ongoing challenges and research opportunities, in: Service-Oriented Computing and Applications (SOCA), 2014 IEEE 7th International Conference on, IEEE, 2014, pp. 230–234.

[7] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, D. Qiu, Security of the internet of things: Perspectives and challenges, Wireless Networks 20 (8) (2014) 2481–2501.

[8] Owasp internet of things project - owasp, `https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project`, (Accessed on 11/18/2017).

[9] C. Kolias, G. Kambourakis, A. Stavrou, J. Voas, Ddos in the iot: Mirai and other botnets, Computer 50 (7) (2017) 80–84.

[10] M. U. Farooq, M. Waseem, A. Khairi, S. Mazhar, A critical analysis on the security concerns of internet of things (iot), International Journal of Computer Applications 111 (7).

[11] Cisco, Securing the internet of things: A proposed framework - cisco, `https://www.cisco.com/c/en/us/about/security-center/secure-iot-proposed-framework.html`, (Accessed on 11/01/2017).

[12] M. S. Farash, M. Turkanović, S. Kumari, M. Hölbl, An efficient user authentication and key agreement scheme for heterogeneous wireless sensor network tailored for the internet of things environment, Ad Hoc Networks 36 (2016) 152–176.

[13] Y. L. Zhao, Research on data security technology in internet of things, in: Applied Mechanics and Materials, Vol. 433, Trans Tech Publ, 2013, pp. 1752–1755.

[14] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, G. Carle, A dtls based end-to-end security architecture for the internet of things with two-way authentication, in: Local Computer Networks Workshops (LCN Workshops), 2012 IEEE 37th Conference on, IEEE, 2012, pp. 956–963.

[15] D. Liu, P. Ning, R. Li, Establishing pairwise keys in distributed sensor networks, ACM Transactions on Information and System Security (TISSEC) 8 (1) (2005) 41–77.

[16] H. Pranata, R. Athauda, G. Skinner, Securing and governing access in ad-hoc networks of internet of things, in: Proceedings of the IASTED International Conference on Engineering and Applied Science, EAS, 2012, pp. 84–90.

[17] N. Ye, Y. Zhu, R.-c. Wang, Q.-m. Lin, An efficient authentication and access control scheme for perception layer of internet of things, Institute of Electrical and Electronics Engineers, 2014-07.

[18] J. Ma, Y. Guo, J. Ma, J. Xiong, T. Zhang, A hierarchical access control scheme for perceptual layer of iot, jisuanji yanjiu yu fazhan/comput, Res. Dev 50 (6) (2013) 1267–1275.

[19] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, C. Xu, Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things, in: Proceedings of the 14th ACM Workshop on Hot Topics in Networks, ACM, 2015, p. 5.

[20] S. Yin, Y. Lu, Y. Li, Design and implementation of iot centralized management model with linkage policy (2015).

[21] M.-M. Moazzami, G. Xing, D. Mashima, W.-P. Chen, U. Herberg, Spot: A smartphone-based platform to tackle heterogeneity in smart-home iot systems, in: Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on, IEEE, 2016, pp. 514–519.

[22] N. Saxena, A. Roy, B. J. Sahu, H. Kim, Efficient iot gateway over 5g wireless: A new design with prototype and implementation results, IEEE Communications Magazine 55 (2) (2017) 97–105.

[23] A. Glória, F. Cercas, N. Souto, Design and implementation of an iot gateway to create smart environments, Procedia Computer Science 109 (2017) 568–575.

[24] C. Razafimandimby, V. Loscrí, A. M. Vegni, A. Neri, A bayesian and smart gateway based communication for noisy iot scenario, in: Computing, Networking and Communications (ICNC), 2017 International Conference on, IEEE, 2017, pp. 481–485.

[25] A. Alshehri, R. Sandhu, Access control models for cloud-enabled internet of things: A proposed architecture and research agenda, in: Collaboration and Internet Computing (CIC), 2016 IEEE 2nd International Conference on, IEEE, 2016, pp. 530–538.

[26] Learn how ifttt works - ifttt, `https://ifttt.com/`, (Accessed on 11/03/2017).

[27] Homeos: Enabling smarter homes for everyone - microsoft research, `https://www.microsoft.com/en-us/research/project/homeos-enabling-smarter-homes-for-everyone/`, (Accessed on 11/03/2017).

[28] openhab, https://www.openhab.org/, (Accessed on 11/03/2017).

[29] Meet the nest learning thermostat — nest, https://nest.com/uk/thermostats/nest-learning-thermostat/overview/, (Accessed on 11/03/2017).

[30] ios - home - apple (uk), https://www.apple.com/uk/ios/home/, (Accessed on 11/03/2017).

[31] S. Guoqiang, C. Yanming, Z. Chao, Z. Yanxu, Design and implementation of a smart iot gateway, in: Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, IEEE, 2013, pp. 720–723.

[32] Pubnub: Making realtime innovation simple — pubnub, https://www.pubnub.com/?utm_source=PayPerClick&utm_medium=Google-Adwords&utm_campaign=PPC-CY16-Q4-Brand-Google-Adwords-Jan-3&gclid=CjwKCAjw7frPBRBVEiwAuDf_LRgXou0MCqdsRCIHq8fWc-4lNBT1-NPe1A9PUd_fc0WpkNJb4ITiMhoCQ50QAvD_BwE, (Accessed on 11/05/2017).

[33] Publish-subscribe (pub/sub) — pubnub, https://www.pubnub.com/learn/glossary/what-is-publish-subscribe/, (Accessed on 11/05/2017).

[34] dweet.io - share your thing- like it ain't no thang., https://dweet.io/, (Accessed on 11/14/2017).

[35] Eclipse kura ^{TM} - open source framework for iot, https://www.eclipse.org/kura/, (Accessed on 11/14/2017).

[36] Zetta api · rest for iot, http://www.zettaapi.org/, (Accessed on 11/14/2017).

[37] S. Brostoff, M. A. Sasse, "ten strikes and you're out": Increasing the number of login attempts can improve password usability (2003).

[38] B. Whitham, Canary files: generating fake files to detect critical data loss from complex computer networks, in: The Second International Conference on Cyber Security, Cyber Peacefare and Digital Forensic (CyberSec2013), The Society of Digital Information and Wireless Communication, 2013, pp. 170–179.

[39] Category:attack - owasp, https://www.owasp.org/index.php/Category:Attack, (Accessed on 12/20/2017).

[40] airodump-ng [aircrack-ng], https://www.aircrack-ng.org/doku.php?id=airodump-ng, (Accessed on 11/16/2017).

[41] Nmap: the network mapper - free security scanner, https://nmap.org/, (Accessed on 11/16/2017).

[42] deauthentication [aircrack-ng], https://www.aircrack-ng.org/doku.php?id=deauthentication, (Accessed on 05/29/2018).

[43] Denial of service attack : Lesson for life - kali linux hacking tutorials, http://www.kalitutorials.net/2014/05/denial-of-service-attack-lesson-for-life.html, (Accessed on 05/29/2018).

[44] aireplay-ng [aircrack-ng], https://www.aircrack-ng.org/doku.php?id=aireplay-ng, (Accessed on 05/29/2018).

[45] Arp spoofing — veracode, https://www.veracode.com/security/arp-spoofing, (Accessed on 12/29/2017).