iscte

INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Towards Transparent and Secure IoT:
Device Intents Declaration, and User Privacy Self-Awareness and Control

João Pedro Santos Lola

Master's Degree in Telecommunications and Computer Engineering

Supervisor:

PhD Carlos José Corredoura Serrão, Associate Professor,

ISCTE - Instituto Universitário de Lisboa

November, 2021

iscte

TECNOLOGIAS
E ARQUITETURA

Department of Information Sciences and Technologies

Towards Transparent and Secure IoT:
Device Intents Declaration, and User Privacy Self-Awareness and Control

João Pedro Santos Lola

Master's in Telecommunications and Computer Engineering

Supervisor:

PhD Carlos José Corredoura Serrão, Associate Professor,

ISCTE - Instituto Universitário de Lisboa

November, 2021

Dedication

It is with genuine gratitude and warm regard that I dedicate this work to the Truphone R&D team and Professor Carlos Serrão. Their support was essential to the work that will be presented in this dissertation and I am very grateful for their guidance every step of the way, ensuring that I was always on the right track.

I would also like to thank my family for the support they showed throughout this time; they made me see the light at the end of the tunnel, and for always pushing me to go further and not to quit when I wanted to give up my master’s degree studies and devote my time and attention to professional work.

Thanks

To my family for their unconditional support.

To Professor Carlos Serrão for his availability, motivation and shared knowledge.

To João Casal and the entire R&D team for their availability, motivation and shared knowledge during the six months of the internship I did at Truphone.

A sincere thank you to everyone.

Resumo

Nos últimos anos, assistimos a uma onda de crescimento da integração de novas tecnologias IoT (*Internet Of Things*) na sociedade. A integração massiva destas tecnologias levou ao aparecimento de vários aspetos críticos que, consequentemente, criou novos desafios, para os quais ainda não foram dadas respostas óbvias. Um dos principais desafios diz respeito à segurança e privacidade da informação dos dispositivos IoT presentes no nosso dia-a-dia. Atualmente, não existem quaisquer garantias por parte dos fabricantes destes equipamentos IoT, que estão conectados nas nossas redes, relativamente à recolha e envio de informação pessoal realizada pelos mesmos, bem como um comportamento expectável.

Assim, neste trabalho, desenvolvemos e testamos uma solução que cujo objetivo é aumentar a privacidade e segurança da informação em redes de dispositivos IoT, na perspetiva do controlo da comunicação dos dispositivos inteligentes na rede. Para incluir-se uma ferramenta capaz de efetuar análise dos pacotes enviados pelos dispositivos IoT e uma outra capaz de definir e permitir a aplicação de regras de controlo de tráfego de rede aos pacotes mencionados. Estas ferramentas foram indispensáveis para a investigação dos dois aspetos centrais desta dissertação, que são a investigação de como as declarações de intenções de comunicação dos dispositivos IoT especificados pelos fabricantes são utilizadas, para facilitarem o controlo de comunicação destes pelos consumidores e permitir-lhes detetar violações dessas intenções e como atribuir ao utilizador/consumidor controlo sobre a comunicação IoT, para que este possa explicitar o pretende e não pretende que os seus dispositivos comuniquem.

Palavras-Chave: Segurança; Privacidade; Redes IoT; Declaração de intenções; Direitos e permissões de comunicação; Análise de tráfego

Abstract

In recent years, we have seen a growing wave of integration of new IoT (Internet of Things) technologies into society. The massive integration of these technologies has led to the emergence of several critical issues which have consequently created new challenges, for which no obvious answers have yet been found. One of the main challenges has to do with the security and privacy of information processed by IoT devices present in our daily life. At present there are no guarantees from the manufacturers of such IoT devices, which are connected on our networks, as regards the collection and sending of personal information, nor an expected behavior.

Thus, in this work, we developed and tested a solution that aims to increase the privacy and security of information in Networks of IoT devices, from the perspective of controlling the communication of smart devices on the network. To include one tool capable of analyzing packets sent by IoT devices and another capable of defining and allowing the application of network traffic control rules to the packets in question. These tools were indispensable for investigation of the two central aspects of this dissertation, which are investigating how the declarations of communication intentions of the IoT devices specified by the manufacturers are used, in order to facilitate control of communication by consumers and enable them to detect violations of those intentions, and how to give users/consumers control over IoT communication, so that they can define what they do and do not want their devices to communicate.

Keywords: Security; Privacy; IoT Networks; Intent declaration; Communication rights and permissions; Traffic analysis

Index

Thanks i

Resumo iii

Abstract v

Index vii

Index of Figures xi

Index of Tables xiii

List of Abbreviations and Acronyms xv

Chapter 1 – Introduction 1

1.1. Context 1

1.2. Research Question 3

1.3. Research Goals 3

1.4. Methodology Approach 4

1.5. Dissertation Structure and Organization 6

Chapter 2 - Literature Review 7

2.1. Network Based Rules Enforcement 7

2.2. Content Filtering & Blocking 12

2.3. Device Communication Permissions 17

2.4. Rights Expression Techniques 20

2.4.1. Yara Rules 20

2.4.2. Open Digital Rights Language 21

Chapter 3 - System Architecture and Workflow 23

3.1. System Architecture 23

3.1.1. Intents & User Rule Management REST API 28

3.1.2. Translator (Integration layer with filtering tool) 29

3.1.3. Snort IPS 30

3.1.4. Web Management Interface 31

3.2. Snort Implementation Workflow 33

Chapter 4 – Tests and Evaluation 37

4.1. Test System Deployment 37

4.2. Functional Tests 40

4.2.1. Description of Test Scenario I – Violations of the manifest of intents 41

4.2.2. Test Scenario I - Results Presentation 43

4.2.3. Test Scenario I - Discussion of Results 44

4.2.4. Description of Test Scenario II - Violations of user-defined rules 45

4.2.5. Test Scenario II – Results Presentation 46

4.2.6. Test Scenario II - Discussion of Results 47

4.3. System Performance 48

4.3.1. Intents Scenario – Results Presentation 49

4.3.2. Intent Scenario – Discussion of Results 53

4.3.3. Rules Scenario – Results Presentation 54

4.3.4. Rules Scenario – Discussion of Results 58
Chapter 5 - Conclusions and Future Work 59
5.1. Main Conclusions 59
5.2. Future Work 60
Bibliography 61
Appendices 67
Appendix A – PCRF and PCEF Solutions Comparison Table 67
Appendix B – ODRL Code Examples 69
Appendix C – Intents & User Rule Management REST API Documentation 70
Appendix D – Translator REST API Documentation 77

Index of Figures

FIGURE 1 - DESIGN SCIENCE RESEARCH METHODOLOGY PROCESS MODEL (PEFFERS ET AL., 2007)........ 5

FIGURE 2 - SIMPLIFIED OPEN5GS ARCHITECTURE 9

FIGURE 3 - SIMPLIFIED MAGMA ARCHITECTURE........ 10

FIGURE 4 - METHODOLOGY FOR DPI USING SNORT (HAFEEZ,2016)........ 14

FIGURE 5 - MANIFEST STATEMENT TO SEND SMS MESSAGES (GOOGLE, 2021) 17

FIGURE 6- INITIAL PERMISSION DIALOG (LEFT) AND SECONDARY PERMISSION REQUEST WITH OPTION TO DISABLE OTHER REQUESTS (RIGHT)(GOOGLE, 2021)........ 18

FIGURE 7- MANIFEST PERMISSION REQUEST (GOOGLE, 2021) 19

FIGURE 8- MANIFEST STATEMENTS TO ACCESS OPERATING SYSTEM RESOURCES (GOOGLE, 2021) 19

FIGURE 9 - YARA RULE EXAMPLE (NAIK ET AL., 2020)........ 20

FIGURE 10 - ODRL INFORMATION MODEL (IANNELLA, 2004) 22

FIGURE 11 - SYSTEM HIGH-LEVEL ARCHITECTURE 24

FIGURE 12 - SNORT LAUNCH COMMAND........ 30

FIGURE 13 - INTENTS & RULES LIST TABLE 31

FIGURE 14 – INTENT & RULE INSERT/UPDATE FORM 32

FIGURE 15 – DOWNLOAD/UPDATE FORM 33

FIGURE 16 – INTENT SCENARIO - SYSTEM WORKFLOW........ 34

FIGURE 17 – RULE SCENARIO - SYSTEM WORKFLOW........ 35

FIGURE 18 – SOLUTION TESTBED ARCHITECTURE 37

FIGURE 19 -TRAFFIC GENERATOR (VM A) NETWORK ADAPTER 39

FIGURE 20 – TRANSLATOR AND SNORT IPS (VM B) NETWORK ADAPTERS 39

FIGURE 21 - INTENT VIOLATION ALERTS 43

FIGURE 22 - RULE DROP PACKET ALERT 46

FIGURE 23 - 1000 PACKETS RUNTIME DATA 49

FIGURE 24 - 5000 PACKETS RUNTIME DATA 51

FIGURE 25 - 10000 PACKETS RUNTIME DATA 52

FIGURE 26 - 1000 PACKETS RUNTIME DATA 54

FIGURE 27 - 5000 PACKETS RUNTIME DATA 56

FIGURE 28 - 10000 PACKETS RUNTIME DATA 57

FIGURE 29 - DELEGATION AS TRANSFER (KEBEDE ET AL., 2021) 69

FIGURE 30 - DELEGATION AS S GRANTING CONDITIONAL USAGE (KEBEDE ET AL., 2021) 69

FIGURE 31 - DELEGATION AS NESTED GRANTING OF CONDITIONAL USAGE (KEBEDE ET AL., 2021) 69

Index of Tables

TABLE 1 - TESTBED SYSTEM SPECS 38

TABLE 2 - TEST INTENTS 41

TABLE 3 - CONTENT TEST INTENT PACKET 42

TABLE 4 - DESTINATION TEST INTENT PACKET 42

TABLE 5 - TEST RULES 45

TABLE 6 - CONTENT TEST RULE PACKET 46

TABLE 7 - 1000 PACKETS VARIABLE DATA 50

TABLE 8 - 5000 PACKETS VARIABLE DATA 51

TABLE 9 - 10000 PACKETS VARIABLE DATA 53

TABLE 10 - INTENT SCENARIO RESULTS SUMMARY 54

TABLE 11 - 1000 PACKETS VARIABLE DATA 55

TABLE 12 - 5000 PACKETS VARIABLE DATA 56

TABLE 13 - 10000 PACKETS VARIABLE DATA 57

TABLE 14 - RULE TEST SCENARIO RESULTS SUMMARY 58

List of Abbreviations and Acronyms

AAA - Authentication, Authorization and Accounting

AGW – Access Gateway

API – Application Programming Interface

ARP – Address Resolution Protocol

DAQ – Data Acquisition Library

DPI – Deep Packet Inspection

EPC - Evolve Packet Core

FTP – File Transfer Protocol

HSS – Home Subscriber Server

HTTP/HTTPS - Hypertext Transfer Protocol / Hypertext Transfer Protocol Secure

IDS – Intrusion Detection System

IMEI – International Mobile Equipment Identity

IMSI - International Mobile Subscriber Identity

IoT – Internet of Things

IP – Internet Protocol

IPS – Intrusion Prevention System

ISP – Internet Service Provider

LTE – Long Term Evolution

MIME - Multipurpose Internet Mail Extensions

MME - Mobility Management Entity

MNO - Mobile Network Operator

NIDPS - Network Intrusion Detection and Prevention Systems

NIDS – Network Intrusion Detection System

ODRL - Open Digital Rights Language

OSI – Open System Interconnection

PCEF – Policy and Charging Enforcement Function

PCRF – Policy and Charging Rules Function

PDU - Protocol Data Unit

PGW – Packet Gateway

QoS – Quality of Service

REL - Rights Expression Language

RTSP – Real Time Streaming Protocol

SIP - Session Initiation Protocol

SMF/PGW-C – Session Management Function/Packet Gateway Control Plane

SMTP – Simple Mail Transfer Protocol

SSH – Secure Socket Shell

TAC – Type Allocation Code

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

UPF/PGW-U – User Plane Function/Packet Gateway User Plane

Chapter 1 – Introduction

1.1. Context

Privacy and security of information are issues when we discuss any Internet-related technology, as they represent the two most fundamental elements in establishing a level of trust between companies and their users. Privacy can be related to any rights a person possesses to control their personal information and how it is used, while security refers to preventing and reducing the probability of unauthorized access, use, disclosure, disruption, modification, inspection, recording or destruction of personal or sensitive information (Pelteret & Ophoff, 2016). Although it is possible to have security without privacy, we can't have privacy without security.

If an organization does not possess strong data security, it can represent a significant differentiating factor for many users and result in loss of trust, as it can lead to potential risks of compromising personal identifiable information, personal health information, medical records, banking information, intellectual property, social security numbers, insurance information and other types of data. If data security is not taken seriously enough, this can result in permanent damage to a company's reputation in the event of a public, high-profile breach or hack trumpeted in the media. Another important consideration are the financial and logistical consequences of a data breach. It is therefore important that information security systems enforce three important principles, namely Confidentiality, which means the protection of information against unauthorized disclosure, Integrity, which means the protection of information against unauthorized modification and, finally, Availability, which means the protection of information against unauthorized destruction and ensuring data is accessible when needed (Pelteret & Ophoff, 2016).

Information of that nature also provides companies and governments with control over an individual, because the more someone knows about a person, the more power they can have over that person. Such information can be used to affect our reputations, influence our decisions and shape our behavior, so if it falls into the wrong hands great harm can be done to us. Therefore, privacy is what keeps companies and governments from having power over people and removing their autonomy, freedom of thought and speech (Pelteret & Ophoff, 2016).

The rapid growth of IoT devices connected to the Internet is causing problems in the market due to privacy and security of information concerns, as reported in several studies (Sivaraman, Gharakheili, Vishwanath, Boreli, & Mehani, 2015), because many of them lack communication security at the end of the communication channel with other systems to prevent against security infringements and data leaks, and a detailed understanding of how they communicate with the network. Looking at the literature, it is possible to find multiple examples of how IoT devices are disrupting security and privacy, such as, for instance, Amazon's Alexa sending private recorded conversations to random numbers on the owner's list of contacts (D. Lee, 2018); Google Nest IP cameras, doorbells and thermostats that could be hacked by third parties (Sears, 2019); Google Home Hub capturing footage from Xiaomi Mijia surveillance cameras from other dwellings (Coble, 2020); and Samsung Smart TVs, which had a vulnerability that allowed third parties to view everything that was going on in the room where the TV was located via the embedded camera and microphone (Chakraborty, 2017).

This has generated a wave of criticism of the lack of transparency among IoT device manufacturers with regard to customer security and privacy. One example of this was Google's failure to disclose in the specifications of its Nest Guard product that the device had a built-in microphone (Blum, 2019). Another example was Amazon's revelation that, in addition to artificial intelligence algorithms, their employees listened to samples of recordings of Amazon Echo devices to improve quality of service (Statt, 2019).

Recent inquiries highlight current consumer concerns about lack of transparency and security and privacy issues in the IoT ecosystem, with respect to gathering and misuse of personal information by the devices and the lack of good quality, reliable consumer information from the organizations about the devices' terms and conditions and privacy disclaimers (Thorun, Vetter, Reisch, & Zimmer, 2017) (CIGI-IPSOS global survey, 2019).

A study conducted by Mozilla revealed that 45% of the 190,000 participants surveyed considered that privacy was the biggest concern when it came to the use of IoT devices and 34.5% of them considered that was up to them to keep connected IoT devices secure and private, versus 34% who considered that it was up to manufacturers to ensure the privacy and security of their devices (Caltrider, 2017). This study shows that consumers do not really trust manufacturers to keep their devices secure and feel it is up to them to safeguard their own privacy and security, which exemplifies the main issue that we have identified in this section.

In another study conducted by the Economist Intelligence Unit, global leader in business intelligence, 92% of the 1629 IoT consumers surveyed said that privacy was their main concern and wished to be informed when their personal information was being collected by IoT devices and to have greater control over the information collected and transmitted (McCauley & Lara, 2018).

To address this problem, this work will focus on developing a proof of concept to ensure transparency of information transmitted by consumers' IoT devices on networks, providing them with control over how they communicate and their behavior, to achieve higher levels of privacy and security.

The work was carried out in the context of the Truphone (www.truphone.com) IoT service provider, which supported the dissertation, providing the necessary context and support to help achieve the stated objectives.

1.2. Research Question

In the context of this work, we are seeking to obtain answers to the following research question:

- How to design and implement a system capable of ensuring transparency in the communication intentions of IoT devices, and user control over data privacy and security when using IoT devices?

1.3. Research Goals

To work towards an answer to the research question identified above, it will be necessary to study and model the statements of communication intentions of IoT devices, i.e., through formulation of how to model what the device will transmit to the network (temperature, brightness, sound, video, among many other things). At the same time, it will also be necessary to study the applicability of rights and permissions techniques to user data and which security policies will govern them.

On the other hand, in order to control communication, it will be necessary to study and test traffic analysis tools capable of identifying various types of communication used by traditional IoT devices.

The main research goals of this dissertation will be:

- To research and develop a system to enable IoT device makers to declare their devices' intentions of communication and a process for verifying the devices' communication compliance with the intentions declared.
- To research and develop a tool that allows users/consumers to control their IoT device traffic, allowing them to define rules to manage the privacy and security of the corresponding data.

1.4. Methodology Approach

The research method used in this dissertation is the Design Science Research methodology (Peffers et al., 2007), which consists of six distinct phases, as illustrated in Figure 1:

1. *Identification of the Problem and Motivation* - Sets out the target problem of the study and explains the importance of resolving it.
2. *Definition of the objectives of the solution* - Sets out the objectives of the solution after defining the problem, to know what is possible and feasible.
3. *Design and Development* - This activity involves determining the functional requirements and architecture of the solution, and then proceeding with its creation.
4. *Demonstration* – Show how the solution developed can solve one or more instances of the problem through experiments, simulations, case studies or other appropriate activities.
5. *Evaluation* - Observe and measure how the solution meets the desired objectives, by comparing the initial objectives of the solution with the observed results. This implies testing the built solution.
6. *Communication* - Writing a scientific paper to communicate the problem and the importance of resolving it.

In this thesis, our initial approach, corresponding to step 1 to step 4, was described in a sequential pattern. *Identification of the Problem and Motivation* and *Definition of the objectives of the solution* are described in section *1.1. Context* and *1.3. Research Goals,* respectively. *Design and Development* are described in *Chapter 3 - System Architecture and Workflow*. The *Demonstration and Evaluation* steps are described in *Chapter 4 – Tests and Evaluation*. The last step, we evaluate the final version and move on to the *Communication* phase which consisted of writing this proposal.

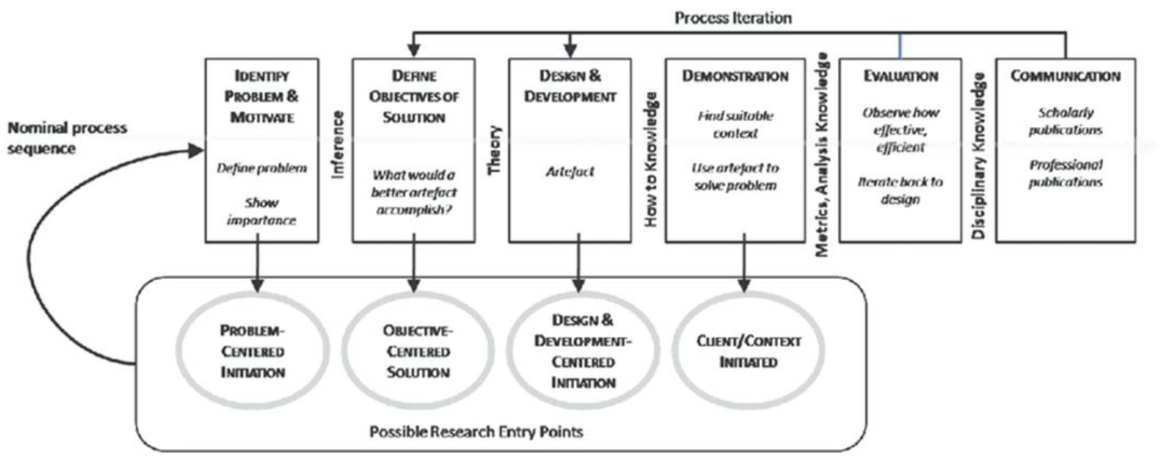

Process Iteration
Nominal process sequence
Identify Problem & Motivate
Define problem
Show importance
Inference
Define Objectives of Solution
What would a better artefact accomplish?
Theory
Design & Development
Artefact
How to Knowledge
Demonstration
Find suitable context
Use artefact to solve problem
Metrics, Analysis Knowledge
Evaluation
Observe how effective, efficient
Iterate back to design
Disciplinary Knowledge
Communication
Scholarly publications
Professional publications
Problem-Centered Initiation
Objective-Centered Solution
Design & Development-Centered Initiation
Client/Context Initiated
Possible Research Entry Points

Figure 1 - Design Science Research Methodology Process Model (Peffers et al., 2007)

1.5. Dissertation Structure and Organization

This dissertation is organized into five chapters. The first chapter introduces the motivation and theme of the investigation in this dissertation, defines the different objectives and presents the structure of the work to be undertaken.

The second chapter reflects the literature review, providing an overview of the most recent and relevant topics in the context of this dissertation.

The subsequent chapter is devoted to design, specification, and implementation of the proposal, which will address the issue of the research previously identified.

The fourth chapter describes how the proposal will be evaluated. It also discusses in a comprehensive manner the various results of the evaluation.

Finally, the last chapter of this work presents the conclusions and provides some directions for future work.

Chapter 2 - Literature Review

In this chapter, we present a literature review on the research subjects wherein we list some of the key issues and topics that will be relevant for this project, to pursue the research objectives stipulated in the previous chapter. These concepts will be crucial for the research, as they will enable the necessary experiments to be carried out to produce results capable of satisfying the requirements that are the object of this research.

To better understand each concept and topic, we have chosen to divide them into sections, which we describe below, explaining how each of them contributes to the objectives.

2.1. Network Based Rules Enforcement

In this section, we present a set of network traffic control functions called PCRF (Policy and Charging Rules Function) and PCEF (Policy and Charging Enforcement Function). These functions are relevant because they allow automatic creation of network traffic control rules and enforcement thereof, which enables control of network traffic (permitted or not) during communication by IoT devices, based on device and content types. These functions are important components of an IoT network, as they represent management and control of the network service. They ensure that QoS (Quality of service) rules and regulations are applied to network packets data, a very valuable aspect of any network policy and security management system.

PCRF represents a node that works in real time to make policy decision control decisions to apply traffic control rules on an IoT network. It is a component which plays a central role in communication networks, operating at the core of the network, to access subscribers' databases and other specialized functions centrally. As it operates in real time, PCRF has a broader strategic significance and potential compared to other policy engines (Li, Brouard, & Cai, 2012).

As part of the network architecture, PCRF aggregates information that enters and leaves the network, supports the system and other sources in real time, through the automatic creation of rules, which are then automatically applied to each of the subscribers who are active on the network (Li et al., 2012).

Its main functions are:

- Real-time management of network and subscriber policy.
- Efficient and dynamic management of network traffic to set priorities for certain types of traffic.
- To define subscriber context based on device type, network, location, and billing information.
- Quality of service assurance and bandwidth management.

PCEF is responsible for the enforcement of rules configured statically or dynamically by PCRF, providing access and information from subscribers to the PCRF through a secure communication interface. Located at the network gateway, PCEF controls traffic through the appropriate inspection of IoT device network packets to ensure that IoT device traffic flows are handled according to network policies defined in the PCRF (Li et al., 2012).

Open5gs

Open5gs (S. Lee, 2020) is a C-Language Open-Source implementation for a 5G Core and EPC (Evolve Packet Core) network that provides PCRF and PCEF for all devices that connect to this network. Subscribers are identified by their IMSI (International Mobile Subscriber Identity), a 15-digit unique number that identifies each device within a network, to which a set of policy control rules are applied specific to each device.

Open5gs is divided into two main planes: the control plane and the user plane. The control plane represents the hub of the core of the solution and contains all the necessary components to manage sessions, mobility, paging and bearers. Those components, as shown in Figure 2, are:

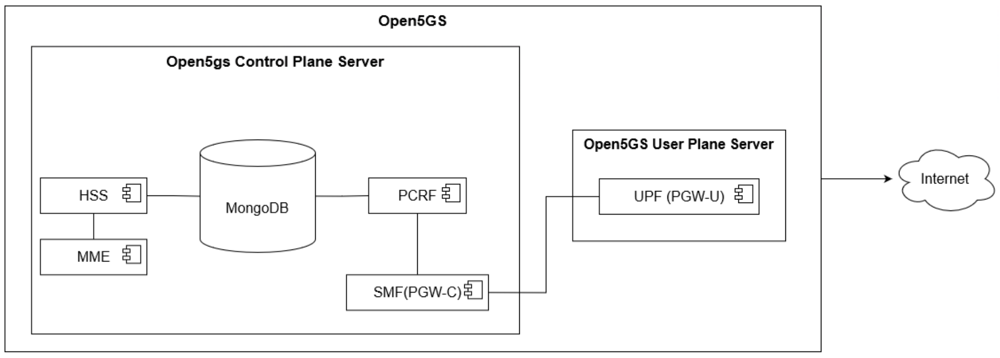


Figure 2 - Simplified Open5Gs architecture

- *MME* – The Mobility Management Entity is the main component that the user equipment communicates with inside a 5G network; the UE has to communicate with MME to connect to the network and to start the procedure for registration on the network. When this happens, the MME runs an authentication procedure providing the UE with a temporary identifier to communicate inside the network; in this way the permanent identifier of the user equipment is not tampered with by anybody else. When the registration procedure finishes, the MME needs to keep temporary data related to the user equipment stored in the HSS, while it stays connected to the network (Rommer et al., 2020).

- *HSS* – The Home Subscriber Server is the master user database that supports the LTE (Long Term Evolution) network. Its main role is to communicate with the network and provide subscriber profile and authentication information, to help with authorization, details of devices, as well as the user’s location and service information. The HSS must provide this information to the MME, to ensure that the LTE network works with legacy or concurrent services (Rommer et al., 2020).

- *SMF/PGW-C* – The Session Management Function or Packet Gateway Control Plane is responsible for setting up connectivity for the user equipment towards the Data Networks as well as managing the user plane for that connectivity. It also manages user sessions, including establishment, modification and release of sessions, and it can allocate IP (Internet Protocol) addresses for the IP PDU (Protocol Data Unit) sessions. Interacts with the PCRF to retrieve policies that are used by the PCEF to enforce on incoming device traffic (Rommer et al., 2020).

- *UPF/PGW-U* – The User Plane Function or Packet Gateway User Plane is responsible for processing and forwarding of user data, which is controlled by the SMF. Acts as an anchor point for the user equipment to allow it to interconnect with external IP networks. The UPF is also responsible for performing various types of processing of the forward data, such as generating charging data records and traffic usage reports. Most importantly in this context, UPF is able to apply packet inspection of the content of packets, for input to policy decisions or basic traffic reporting (Rommer et al., 2020).

Magma Core

Magma (LF Projects LLC, 2021) represents an open-source software platform that gives network operators an open, flexible, and extendable mobile core network solution. Like Open5GS, it also provides PCRF and PCEF for all devices that connect to the network. Subscribers are also identified by their IMSI but, unlike Open5GS, the subscribers are associated with the policies and not the policies with the subscriber. An advantageous feature of Magma is that it makes it possible to create more than one network, with which a different set of policies and subscribers can be associated.

The Magma architecture is divided into three major components, as shown in Figure 3:

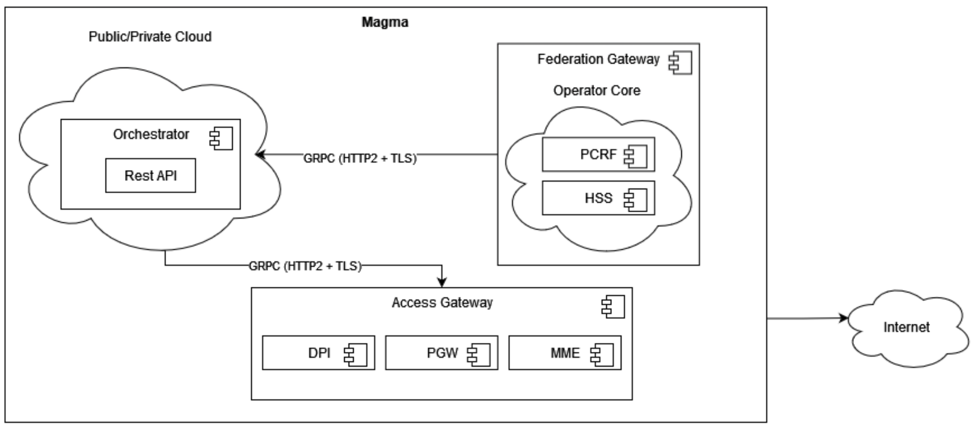


Figure 3 - Simplified Magma architecture

- *Access Gateway (AGW)*: Represents a component that provides network services and policy enforcement. In an LTE network, the AGW implements an EPC, and a combination of an AAA (Authentication, Authorization and Accounting) and a PGW (Packet Gateway). It works with existing, unmodified commercial radio hardware (LF Projects LLC, 2021).

- *Orchestrator*: Represents a cloud service that provides a simple and consistent way to configure and monitor the wireless network securely that can be hosted on a public or private cloud. The metrics acquired through the platform allow the user to see the analytics and traffic flows of the wireless users through the Magma Web user interface frontend that is supported by REST API (Application Programming Interface) backend (LF Projects LLC, 2021).

- *Federation Gateway*: Its main job is integrating the MNO (Mobile Network Operator) core network with Magma using standard 3GPP interfaces to existing MNO components. Also acts as a proxy between the Magma AGW and the operator's network and facilitates core functions, such as authentication, data plans, policy enforcement and charging to stay uniform between an existing MNO network and the expanded network with Magma (LF Projects LLC, 2021).

Both Open5gs and Magma (*Appendix A – PCRF and PCEF Solutions Comparison Table*), represent open-source network connectivity solutions that offer built-in PCRF and PCEF, allowing the possibility of filtering the traffic flows of devices that connect to these networks. For that reason, we can implement a mechanism to ensure the privacy and security of users.

2.2. Content Filtering & Blocking

In this section, we present the DPI (Deep Packet Inspection) technique that is relevant for the analysis of data packets that are exchanged during the communication of IoT devices, which in combination with functions mentioned in section *2.1. Network Based Rules Enforcement* decide whether the packet is complying with its communication manifest, based on its origin, destination, and content, by applying the correct traffic control rules to each of the IoT devices to prevent them from violating the privacy and security of the owner.

DPI is an advanced method for examining and managing network traffic. It is a form of packet filter that finds, identifies, sorts, redirects or blocks packets with specific information compared to conventional packet filtering, which only analyses packet headers, making it unable to detect certain content within packets (El-Maghraby, Elazim, & Bahaa-Eldin, 2018).

The DPI process begins by examining the contents of packets that pass through layers 3 and 4 of the OSI (Open System Interconnection) Model, where the IP and TCP (Transport Control Protocol) protocols are located, respectively. Here it makes a real-time decision whether the packet can continue to its destination, considering the content it carries. That decision is based on the rules assigned to it by the entity that established the initial setup, the ISP (Internet Service Provider) or the network manager. Unlike conventional packet filters, DPI can examine the content of messages and identify the application or service that creates it, offering the ability to program filters to search for or redirect traffic from a specific IP address or online service (El-Maghraby et al., 2018).

An IPS (Intrusion Prevention System) represents a network security/threat prevention technology that examines network traffic flows to detect and prevent vulnerability exploits, identifying problems with security policies and deterring individuals from violating said security policies. Unlike an IDS (Intrusion Detection System), a passive system that scans traffic and reports back on threats, the IPS is placed inline, in the direct communication path between the source and destination, to actively analyze and take automated actions on all traffic flows that enter the network.

Some of those actions are (Lakshminarayana, Philips, & Tabrizi, 2019):

- Sending alarms.
- Dropping the malicious packets.
- Blocking traffic from the source address.
- Resetting the connection.

The IPS detection method can be based on signatures or anomalies. Signatures are based on patterns of well-known network attacks. The signature match process is done by comparing packet flows with the signatures to see if there is a pattern match. An anomaly-based method, however, use heuristics to identify threats, for instance by comparing a sample of traffic against a known baseline. Therefore, it must work efficiently to avoid degrading network performance, because exploits can happen in near real-time. Additionally, it must also detect and respond accurately to eliminate threats and false positives, legitimate packets that are misread as threats (Lakshminarayana et al., 2019).

Deep Packet Inspection using Snort IPS

(Hafeez, 2016) proposed a solution that uses an effective open-source IPS software called Snort (Roesch, 2016), a tool for detecting network traffic whose primary use is intrusion detection in systems. This works by creating rules for network traffic control, having all the features necessary to perform DPI in an entire network, implemented on a single computer. This eliminates the need for expensive devices such as routers or routing services with extra costs to perform this service, offering a low-cost solution with a low degree of implementation.

According to (Bhosale & Mane, 2016), Snort has the advantage of being easy to install on any type of network, possessing a detailed and scrutinized database of signatures, being lightweight and offering the possibility of acting as an IPS. Despite being the most popular IDS/IPS, Snort has the disadvantage of possessing a very large rule database, being expensive for monitoring packets in large networks and of failing to detect fragmented packets at high network speeds (greater than 5 Gbps).

To perform DPI, (Hafeez, 2016) configured Snort in inline NIDS (Network Intrusion Detection System) mode to block and track network traffic. In inline mode, a bridge network is created, through which all network traffic passes through Snort, so that the desired policies can be implemented.

In Figure 4, we can see the methodology used by (Hafeez, 2016) in his project, using Snort for DPI and configuring it to operate in IPS mode.

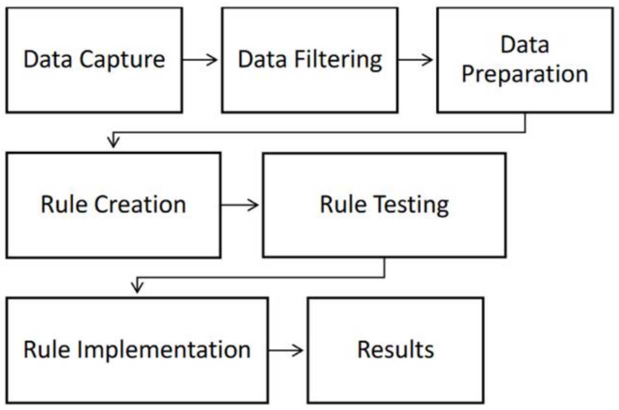


Figure 4 - Methodology for DPI using Snort (Hafeez,2016)

According to (Hafeez, 2016), the first step in creating a rule for a Web application is to get the content of the application by capturing Web traffic from it. For this process, (Hafeez, 2016) used Wireshark (Combs, Ramirez, Harris, & Sharpe, 2016) software to capture traffic at the expense of Snort, simply because it is easier to read packets. Wireshark is an open-source program that allows you to view the details of each packet sent and received on a network in real time. After capturing Web traffic, it is filtered to remove unwanted traffic by using filters to get the desired results.

The purpose of the packet traffic filter is to get traffic to rule creation, which is then stored in the automatically generated packet capture file by Wireshark. With this content, you can create rules so that Snort generates alerts when it finds the same content in network traffic.

During validation of the rules created, if Snort encounters an inconsistency in the rule, it shows an error message and does not validate the rule until it is corrected. When validation is complete, the rules are tested on the packet capture files to see if they work and give the desired results. They are implemented on real traffic to receive alerts only after they are tested.

Another alternative to Snort is open-source software called Suricata (Julien, Jonkman, & Metcalf, 2007) that also can be implemented as an IDS or an IPS. According to (Bhosale & Mane, 2016), the advantages of Suricata are its multithreaded architecture, automatic detection of protocols, NSM (Network Security Monitoring) and the filtering of alerts and events. The main disadvantage of Suricata is its high consumption of system resources.

Deep Packet Inspection using Suricata

(Shah & Issac, 2018) published a study that investigated the performance of Snort and Suricata in accurately detecting malicious traffic on computer networks. In their study, (Shah & Issac, 2018) observed and measured the performance costs on CPU (Central Process Unit), memory utilization and network drop rate of both solutions for 10 Gbps and 20 Gbps network speeds to determine their detection accuracy. They were able to prove that both solutions were efficient and capable of high-performance IDS, each with its own strengths and weaknesses.

Their experimental results show that Snort uses less computational resources to process 10 Gbps worth of network traffic in comparison to Suricata, which required higher computational resources to process network traffic due to the three detection stages built into it. The results also showed that Suricata's multi-threaded architecture possessed a higher ability to process 10 Gbps worth of traffic with minimal packet drop rate in comparison with Snort's single threaded architecture, which proved less efficient for the same load. Furthermore, (Shah & Issac, 2018) test results showed that Snort was able to process only 60866 packets/second, while Suricata processed 82223 packets/second.

According to (Shah & Issac, 2018) both solutions triggered a high rate of false positive alarms, in terms of malicious and legitimate traffic detection accuracy. Snort registered 55.2%, while Suricata triggered 74.3%. (Shah & Issac, 2018) also claim that Snort had high detection of false negative alarms, of 6.7% against Suricata’s 16.7% for the same malicious traffic. (Shah & Issac, 2018) concluded by saying that Snort has proved itself to be the superior IDS in various respects notwithstanding a high rate of false positive alarms, in comparison with Suricata, which is incapable of detecting data link layer traffic.

In his master's thesis in IT security at the Science Faculty of the University of Lisbon, (Calado, 2018) found Suricata to be easier to install, set up and configure than Snort, while also offering highly customizable options, enabling it to be integrated with several external software's and deployable in different types of hardware. Even performing scaling of Snort, he came to the conclusion that he could not reach Suricata's performance in his setup with every tuning technique Snort had to offer.

(Calado, 2018) also highlighted that both these tools provide extensive support documentation with all the necessary answers to the most important questions. Their community-driven approach also means that we can collaborate with other developers to share knowledge and experience and report errors that contribute to improving the solution.

The IPS process will be crucial for the solution that will be implemented, as it will be necessary to inspect the contents of the packets, to ascertain what type of information they carry and how this corresponds to what is specified by the communication intentions of the devices. The IPS technique can be applied at the PCEF level, as presented in section *2.1. Network Based Rules Enforcement*, to analyze the contents of the packets and, based on this, to ask the PCRF which policies are appropriate and applicable to the case in question.

Alternatively, to the PCRF and PCEF solutions present in section *2.1. Network Based Rules Enforcement*, we can implement Snort or Suricata in IPS mode, as described by (Hafeez, 2016), (Shah & Issac, 2018) and (Calado, 2018), which has its own system of rules to control network traffic, based on the information that is contained within the packets it receives and analyses.

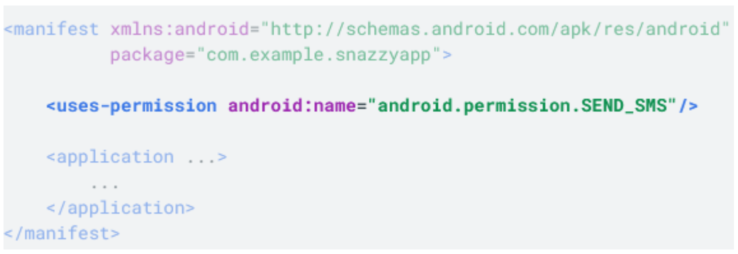
2.3. Device Communication Permissions

In this section, we present how permissions are manifested in the Android operating system, so that apps have access to device resources. Understanding how devices manifest their permissions is relevant to allow us to define traffic control rules using section 2.1. Network Based Rules Enforcement functions, which detect when those permissions are being violated, by analyzing the content using the section 2.2. Content Filtering & Blocking.

Permissions on an Android device are intended to protect privacy. Therefore, applications that can be installed on the operating system need permission to access sensitive data e.g. (Contacts and SMS Messages), as well as system-specific resources e.g. (Camera and Internet). Depending on the resource, the system may automatically grant the necessary permissions, or ask the device user to approve that application's access to the device's resources (Felt, Chin, Hanna, Song, & Wagner, 2011).

One of the most important points of security of the Android operating system architecture is that, by default, no application has permissions to perform any type of operation, which may be harmful to other applications that are installed on the device, to the operating system itself, or even to the device owner. In this context, it includes reading and writing user private data (contacts or emails), reading or writing files from another application, Internet access, keeping the device in the active state, among other actions (Felt et al., 2011).

Applications disclose the permissions required for their execution through their manifest file, using the tag <uses- permission>, as can be seen in Figure 5.



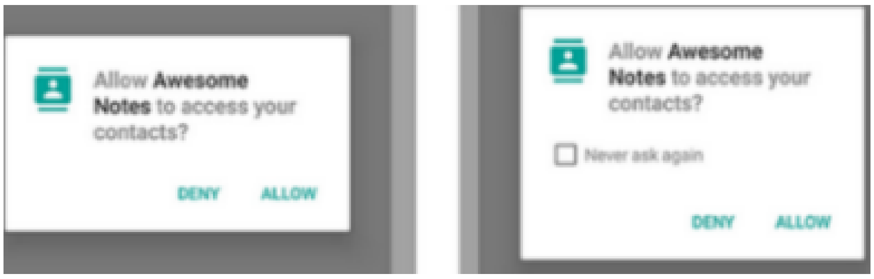
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.snazzyapp">

    <uses-permission android:name="android.permission.SEND_SMS"/>

    <application ...>
        ...
    </application>
</manifest>
```

Figure 5 - Manifest statement to send SMS messages (Google, 2021)

This manifest file is the most important resource for analyzing the communication behavior of components. Through this file, a developer can specify which components will be able to communicate with components from other applications and whether that component can communicate explicitly or implicitly (Jha, Lee, & Lee, 2015).In simple terms, explicitly communication can be referred to expressing something clearly and unambiguously, without leaving any room for confusion, in another hand, implicitly communication can be referred to expressing something indirectly or implied, which may lead to misinterpretation or confusion (Morgan, 2021).

The system, automatically, considers application of permissions that are not identifiable has risks to the invasion of the user's privacy or the operation of the device, in its manifest (Felt et al., 2011). Otherwise, the application lists its required permissions, that can only be granted by the device user, e.g., sending an SMS message, as shown in Figure 5. Therefore, when it comes to permissions classified as dangerous, only the user has the power to grant them. The Android operating system cannot grant any permissions to this type of application.

On devices running the Android 6.0 operating system or newer, the user does not receive any notification about application permissions at the time of installation. The application only interacts with the user for consent to dangerous permissions when the application runs after installation. When the application requests permission from the user, a system dialogue box is shown with two options, Allow or Deny**.** In some cases, the system may require permission again from the user, if the user initially denied it. Secondarily, the user will have the option Never ask again to tell the system that they do not want this permission to be requested again (Felt et al., 2011). Both cases are illustrated in Figure 6.



Allow **Awesome Notes** to access your contacts?

DENY ALLOW

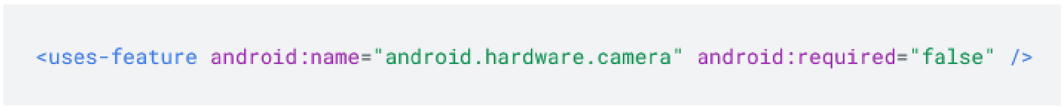
Allow **Awesome Notes** to access your contacts?

Never ask again

DENY ALLOW

Figure 6- Initial permission dialog (left) and secondary permission request with option to disable other requests (right)(Google, 2021)

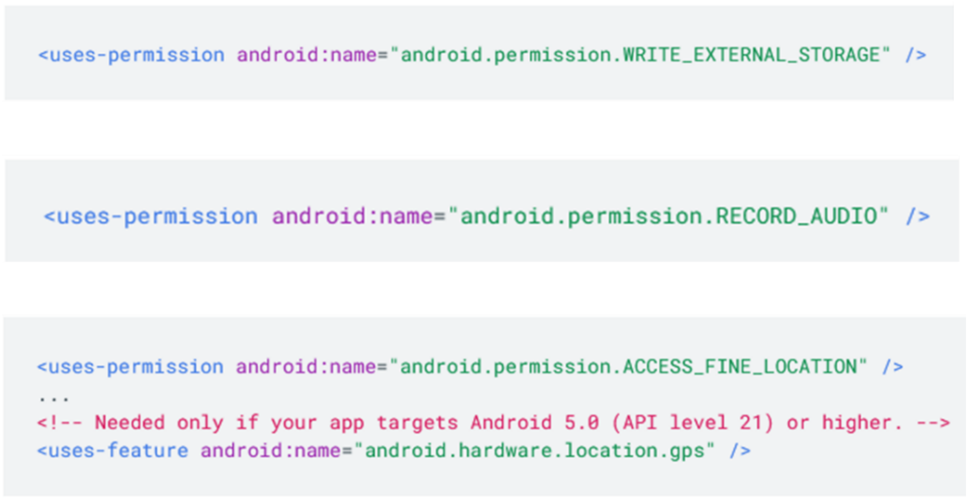
Access to some device hardware features such as Bluetooth or the Camera requires granting permission to the application that will use them. However, not all Android devices have these hardware features. If the application requests permission for the Camera, the <uses-feature tag > must be included in the application manifest to declare those features (Felt et al., 2011), as shown in Figure 7.



```
<uses-feature android:name="android.hardware.camera" android:required="false" />
```

Figure 7- Manifest permission request (Google, 2021)

If we want to obtain permissions to use camera-related features such as storage, audio recording, location access, we must also specify this information in the application manifest via the <uses-permission /> tag (Felt et al., 2011), as shown in Figure 8.



```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
...
<!-- Needed only if your app targets Android 5.0 (API level 21) or higher. -->
<uses-feature android:name="android.hardware.location.gps" />
```

Figure 8- Manifest statements to access operating system resources (Google, 2021)

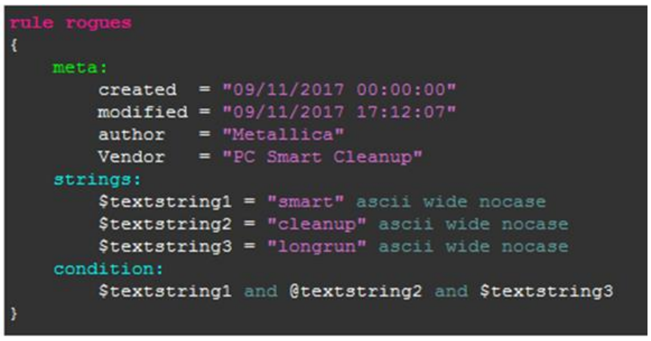
2.4. Rights Expression Techniques

In this section, we present some rights expression techniques that are relevant to specify traffic control rules that are applied to packets issued by devices, which should nevertheless be analyzed using the section *2.2. Content Filtering & Blocking* technique to verify whether they comply with the device permissions specified in section *2.3. Device Communication Permissions*.

2.4.1. Yara Rules

Yara rules are a way to identify malware or other files by creating rules that look for certain characteristics. They are part of Yara software, initially developed by VirusTotal (Quintero et al., 2004), with the intention of helping researchers identify and classify malware samples by building a common security knowledge platform to share among cybersecurity professionals. With this software it is possible to create descriptions of malware families, based on textual and binary patterns (Naik, Jenkins, Cooke, Gillett, & Jin, 2020).

Yara rules are easy to write and understand, because all rules start with the word rule, followed by the name or identifier, as can be seen in Figure 9.



```
rule rogues
{
    meta:
        created   = "09/11/2017 00:00:00"
        modified  = "09/11/2017 17:12:07"
        author    = "Metallica"
        Vendor    = "PC Smart Cleanup"
    strings:
        $textstring1 = "smart" ascii wide nocase
        $textstring2 = "cleanup" ascii wide nocase
        $textstring3 = "longrun" ascii wide nocase
    condition:
        $textstring1 and @textstring2 and $textstring3
}
```

Figure 9 - Yara rule example (Naik et al., 2020)

The rules are composed of two mandatory sections:

- The *strings section* is the part of the rules that is defined; however, this section can be omitted if the rule is not dependent on any string. Each string is identified by the $ symbol, followed by a string of alphanumeric characters and underscores.

- The *conditions section* is where the rule logic resides and is the only section that is required because it contains a Boolean expression that determines when a file or process satisfies the rule or not. Generally, the condition refers to strings previously defined by their identifiers. In this context, the string identifier acts as a Boolean variable to evaluate for true, if the rule was found in the memory of the file or process, or false if otherwise.

2.4.2. Open Digital Rights Language

ODRL (Open Digital Rights Language) is an expression policy language that provides a flexible and interoperable information model, vocabulary, and coding mechanisms to represent statements about the use of content and services. This model describes the underlying concepts, entities and relationships that form the foundation of language policy semantics (Iannella, 2004).

This model consists of the following classes, as seen in Figure 10:

- *Policy* - Non-empty group of permissions, prohibitions, or duties. This class is the parent class of the Set, Offer and Agreement subclasses:
    - *Set* - A policy subclass that supports generic rules.
    - *Offer* – A policy subclass that supports assigned party rule offerings.
    - *Agreement* – A subclass that supports compliance with party rules.
- *Asset* - A resource or collection of resources that are subject to a rule.
- *Party* - An entity or a collection of entities to take roles in rules.
- *Action* - The operation of a resource.
- *Rule* – An abstract concept that represents the common characteristics of permissions, prohibitions, and duties.
    - *Permission* – The ability to act on a resource. This permission may also have the right to express an agreed action, which must be enforced.
    - *Prohibition* – The inability to act on an appeal.

  - *Duty* - The obligation to act.

- *Constraint/ Logical Constraint* - A Boolean expression that redefines an action or group collection or resources applicable to a rule.

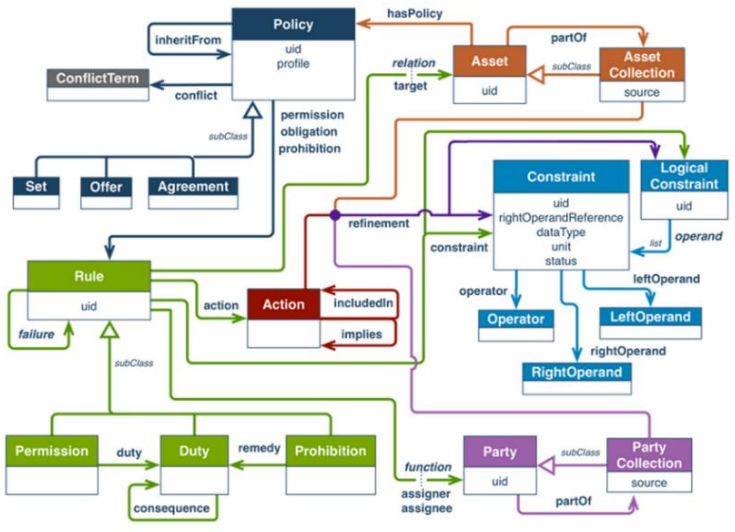


Figure 10 - ODRL information model (Iannella, 2004)

An example of the application of the ODRL, could be an institutional delegation scenario. If suppose that Organization called OrganizationX, located in the Netherlands that maintains a registry of patient data, forms a data-sharing agreement with OrganizationY, an institution in Belgium. The data-sharing agreement grants OrganizationY the permission to acess the data and the possibility of delegating this permission to a third party, OrganizationZ. As a result, the latter will be allowed acess to have access if OrganizationY decides to delegate the permission received from OrganizationX (Kebede, Sileno, & Van Engers, 2021).

This means we can use an ODRL *Agreement* to represent the *Permission* of the transfer of the ownweship of the patient data from OrganizationX to OrganizationY. Since that OrganizationY possess now the ownership of the data, we can assume that ownership includes the possibility of transferring the asset again to someone else (e.g., OrganizationZ). But has a safety measure, OrganizationX can use *grantUse*, to create policies about the target asset, and *nextPolicy*, to indicate the policy applies to a third party, to restraint the use of the data only to a third party and not further (Kebede et al., 2021). Code representation located in Appendix B – ODRL Code Examples.

Chapter 3 - System Architecture and Workflow

In this chapter, we present the prototype that was implemented and the components that ensure the transparency of the information transmitted by consumers' IoT devices in the networks, providing them with control over their devices' communications and ensuring that these devices communicate in accordance with what is stated on their manifests.

3.1. System Architecture

For the present prototype, we decided to create two separate entities, the device manufacturer and the IoT device owner. The device manufacturer traffic control rules are represented by the *Intent rules* and IoT device owner traffic control rules are represented by the *User rules*.

The *Intent rules* represent the intentions and specifications of the manufacturer, to be defined in the device manifest, regarding the device communication requirements, the information the device sensors gather and what is transmitted to the network. These rules will make the communication specifics transparent to anyone using the device and will make it possible to check whether the device is complying with its declared manifest or not. If the device is complying with the manifest intentions, this is considered normal behavior. If not, the IoT device owner will be informed by the system and it will be up to them to decide what to do next (e.g., to block a device from communicating sound to the internet). The device owner is empowered to act upon their device's behavior with the *User rules*.

The *User rules* represent the IoT device owner's intentions and specifications, as regards what they permit the device to send to the Internet. These rules give the owner of the device complete control over it. In this way, the owner can create rules to block the device from sending unwanted information that in their view violates their privacy and security.

Figure 11 represents the overall system architecture made up of the three main services needed to implement the concepts described: Intent & User Rule Management, Translator and Network Traffic Capturing & Filtering. The services referred to are intended to live inside the Truphone core network performing the actions described for both rules.

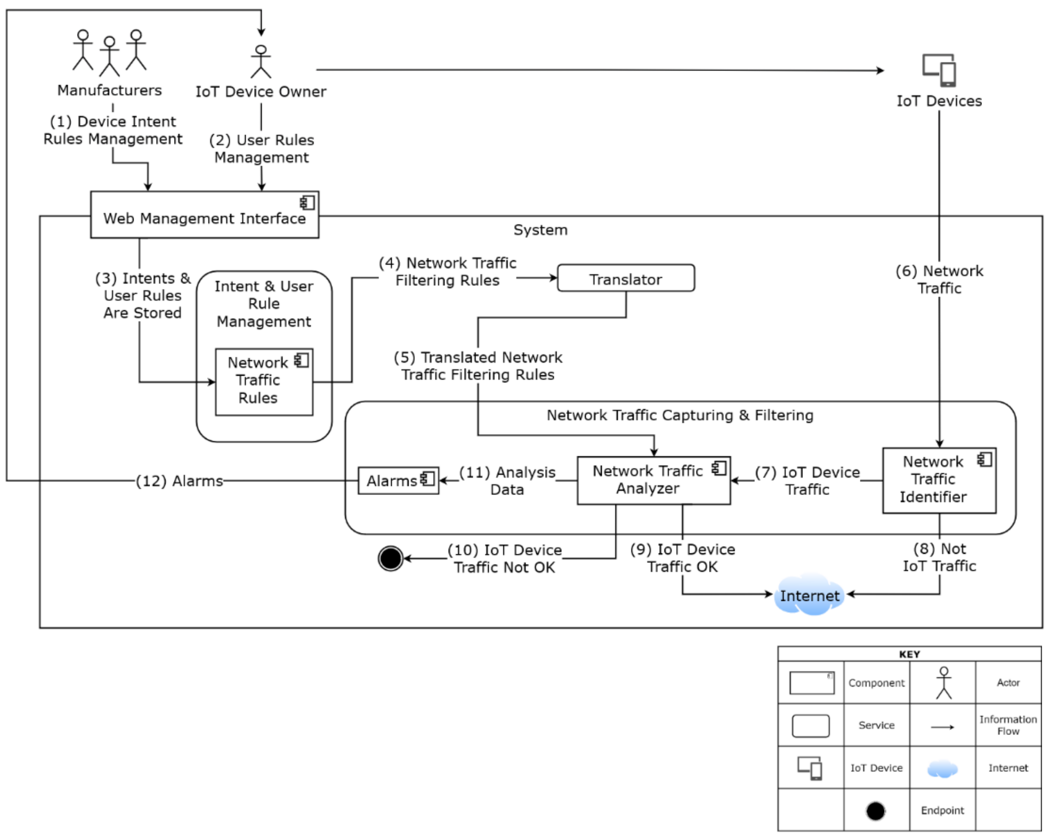

Manufacturers
IoT Device Owner
IoT Devices
(1) Device Intent Rules Management
(2) User Rules Management
Web Management Interface
System
(3) Intents & User Rules Are Stored
Intent & User Rule Management
Network Traffic Rules
(4) Network Traffic Filtering Rules
Translator
(5) Translated Network Traffic Filtering Rules
(6) Network Traffic
Network Traffic Capturing & Filtering
(12) Alarms
Alarms
(11) Analysis Data
Network Traffic Analyzer
(7) IoT Device Traffic
Network Traffic Identifier
(10) IoT Device Traffic Not OK
(9) IoT Device Traffic OK
(8) Not IoT Traffic
Internet
KEY
Component
Actor
Service
Information Flow
IoT Device
Internet
Endpoint

Figure 11 - System high-level architecture

By creating these two types of traffic control rules, we have introduced the concepts of *whitelist* and *blacklist*. The *blacklist* concept is applied to User rules. In this case, we create rules that prevent the packets from reaching their destination to enforce the user's control over their IoT traffic and control over their privacy and security. The *whitelist* concept is applied to Intent rules and aims to describe the rules that are based on the devices' transparent / clear intentions. It is used to learn when devices are not complying with their manifest, by creating rules that trigger specific alarms (e.g., according to the packets' content and destination). We can also create blacklist rules from the whitelist rules if we wish.

The Intent & User Rules Management service consists of two components: the Web management interface, which provides the appropriate mechanisms for IoT device owner and manufacturer to enter, modify and delete user-defined rules and manufacturer intents, respectively. The Network Traffic Rules component represents the backend of the interface, where the CRUD (Create, Read, Update and Delete) database operations of the user rules and manufacturer intents are implemented.

To specify the traffic control rules in the Intents & User Rule Management service, we decided to design our own rules specification, to the detriment of the rights expression methodologies that we presented, in section *2.4. Rights Expression Techniques* of *Chapter 2 - Literature Review*.

The reason we did not opt for the methodologies mentioned was that rights expression methodologies like the one presented in section *2.4.1. Yara Rules*, were designed to detect malware samples, based on text or binary string patterns by matching signatures/strings with existing malware signatures/strings to match against files, folders, or processes. This meant most traffic control systems would not understand these rules directly and translating them into a traffic control system would have been more difficult, because the way they are expressed is not the same as Yara rules. Regarding section *2.4.2. Open Digital Rights Language*, we also came to the conclusion that these were designed to be applied to digital content to protect against copyright infringement and how content is used by third parties, rather than for use in network traffic control. However, these can be extended to add more functionalities but, as regards network traffic control, we could find no available studies that present proofs that the adaptation is relevant to our purpose.

Additionally, using these expression techniques would have made translation to the expression technique used by traffic control rules solutions or others more complex than necessary for the purpose of this work. Our traffic control rules specification in contrast is much more compact, generic, and much easier to understand, which means it can be adapted to the different expression techniques used by some of the solutions presented in sections *2.1. Network Based Rules Enforcement* Open5gs and Magma and *2.2. Content Filtering & Blocking* Deep Packet Inspection using Snort IPS, because it was created from scratch with network traffic control in mind.

The *Translator* represents a service that receives the traffic control rules from the Intent & User Rules Management service and translates them into the expression technique that is used by the Network Traffic Capturing & Filtering service to define traffic control rules. This service is implemented in the same machine that the Network Traffic Capturing & Filtering service is installed on and configured to store the translated rules directly in the rules folder of the Network Traffic Capturing & Filtering service configuration folders.

However, this translation service could have been implemented directly in the Intent & User Rules Management service. But after some research and reflection, we decided to implement it in a complementary micro service to ensure its portability and interoperability with other existing traffic policy control systems.

The Network Traffic Capturing & Filtering service consists of three components: Network Traffic Identifier, Network Traffic Analyzer and Alarms. These components represent the core of Snort IPS that we have installed and configured to perform the IoT devices network traffic control.

For the sake of simplicity and to focus on proving the central concepts of device intention declarations and user controlled IoT privacy and security, we assumed that the content of the packets issued by the IoT devices and captured by the Network Traffic Capturing & Filtering service is transported via HTTP (Hypertext Transfer Protocol) messages and not HTTPS (Hypertext Transfer Protocol Secure) messages, so the content that is parsed is not encrypted.

This option also finds strengths in the research of (Amar et al., 2018), where some of the devices used in their studies revealed that the communication of device states and credentials was done via plain HTTP with other devices and the outside world. Their study also showed that some devices received their software updates via HTTP and that most of this traffic corresponds to requested images and video thumbnails from manufacturer servers. In another study by (Mazhar & Shafiq, 2020), they noted that all smart home IoT devices with the exception of smart cameras access the WEB over HTTP for some portion of traffic, with health & wearables, smart assistants and smart TV corresponding to around 20% of traffic over HTTP.

The Network Traffic Identifier is the component that identifies whether traffic being sent to the Internet has its origin on an IoT device. If so, it is sent to the Network Traffic Analyzer, otherwise, the network traffic goes directly to the Internet. For simulation purposes all the generated IoT traffic that is captured by this component is created by the Ostinato (P, 2010) traffic generator, located before the Network Traffic Capturing & Filtering service. Ostinato simulates the IoT device's behavior when violating the manufacturer manifest communication intents and user-defined rules. The identifier of IoT traffic is contained inside the packet that is captured for analysis and is made up of the TAC (Type Allocation Code), a unique 8-digit code which allows identification of the manufacturer and model of the device individually within a network for the manufacturer intents, and the IMEI (International Mobile Equipment Identity), a unique 15-digit code which acts as a fingerprint of the device, allowing it to be identified within a network for the user-defined rules.

The Network Traffic Analyzer analyses the IoT traffic according to the traffic control rules defined. If the traffic complies with what is specified, it is redirected to the Internet; if not, alarms will be triggered if traffic is violating the manufacturer intent manifest or blocked from reaching the Internet if the traffic is violating the user-defined rules.

The Alarms component receives information about network traffic that complied with the rules and the traffic that did not. This component has the main task of alerting the IoT device owner to the system activity and especially to any device with a behavior not declared by the manufacturer. With this information, the user may choose to block traffic from the device that falls outside the intentions manifest. The information presented to the owner consists of the timestamp of the captured packet, the transport protocol, source and destination addresses, and the content it was carrying that triggered the specific policy rule violation.

Regarding the Network Traffic Capturing & Filtering service, we considered multiple alternative solutions and approaches. The first approach was to use the solution that Truphone has implemented in their core network, based on PCRF and PCEF for network-based rule enforcement. Unfortunately, it was not possible to use that solution without causing major inconvenience to clients who were making use of their service and no alternative (offline) testing environment was available or could be created for research purposes.

Next, we considered available network-based rules enforcement open-source solutions, which led to the two solutions Open5gs and Magma, based on 5G and EPC networks that provide PCRF and PCEF for all devices connected to the network, as presented in section *2.1. Network Based Rules Enforcement*.

In this second approach, we decided to take our study with Open5gs further, because Magma required a paid-for DPI software component, which was not an option for this research project. Open5gs was also found to have some major issues after implementation of the solution, as during the testing phase it was not possible to get the PCRF and PCEF components to talk to each other during operation. This solution was therefore dropped.

For our third approach, we decided to go with the solution Deep Packet Inspection using Snort IPS that was presented in section *2.2. Content Filtering & Blocking*, a tool designed primarily for network traffic intrusion detection, configured to act as an IPS, enabling it to perform DPI on packets captured from the network and apply a set of user-defined rules to block packets or allow them to pass through to the network, as an alternative to the PCRF and PCEF network-based rules enforcement solutions Open5gs and Magma discussed in section *2.1. Network Based Rules Enforcement*.

3.1.1. Intents & User Rule Management REST API

The Intents & User Rule Management REST API, documented in technical format in Appendix C – Intents & User Rule Management REST API Documentation, is designed to allow manufacturers to state their communication intentions and to enable the IoT device owner to create user rules to control the information communicated by the devices (to protect their privacy and ensure information confidentiality). This REST API provides manufacturers and IoT device owners alike with endpoints to allow the intents and user-defined rules to be retrieved from the local database to be displayed. The entry methods allow the manufacturer to enter their device communication intent statements and the IoT device owner to insert their defined rules to control the device communication intents into the database. The update methods allow for modification of existing intents and user-defined rules in the database. Finally, the delete method allows withdrawal of device intents or user-defined rules from the database.

For this REST API, two entities were created to represent the manufacturer intent and the user-defined rules, designated as intent and rule.

An intent is defined by a UUID (Universal Unique Identifier), the *device type*, which is represented by the device TAC. Also part of this intent are its properties as defined by the *content type*, such as Audio, Video, or Text, by the *communication protocol*, which is used to represent the message that contains that content, such as HTTP, RTSP (Real Time Streaming Protocol) or SIP (Session Initiation Protocol), and by the *destination address* of the intent, which is represented by a range of IPs to which the content is to be sent.

A rule has the same fields as the intent, with the exception of the *device identifier* field, which is represented by the device's IMEI.

3.1.2. Translator (Integration layer with filtering tool)

The Translator REST API, documented in technical format in Appendix D – Translator REST API Documentation, is designed to allow the translation of intents and IoT device owner rules that it receives from the Intents & User Rule Management REST API into the expression technique used by Snort. This REST API provides the methods to create Snort-compatible rules from the manufacturer intents and user-defined rules created in the Intents & User Rule Management REST API. The update methods allow modification of the existing Snort rules by replacing them with new ones entered by the manufacturer or IoT device owner. The delete method allows the withdrawal of device Snort rules from the rule files located in the Snort configuration folders. To enable Snort to detect and interpret the content of a packet it has captured, a method has been created to transform the content type that it receives from Intents & User Rule Management REST API service from text format into hexadecimal format when the Snort rule is created.

For this REST API, an additional entity has been created to complement existing ones in Intents & User Rule Management REST API, called a Snort rule. A Snort rule is defined by:

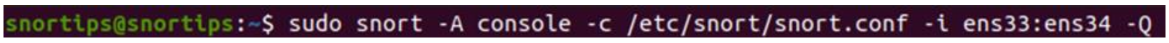
- *Rule Action* - which specifies what action the Snort rule performs on the packet it received, such as *alert, drop, log, pass, reject or drop.*
- *Rule Protocol* - which specifies which protocol should be parsed for suspicious behavior, such as TCP, UDP, ICMP or IP.
- *Source IP* - which specifies which source IP(s) address(es) of the packet.
- *Source Port* - which specifies which source port(s) of the packet.
- *Flow Direction* - which specifies which direction of packet flow, unidirectional (->) or bidirectional (<>).
- *Destination IP* - which specifies the destination address(es) of the packet.
- *Destination Port* - which specifies the destination port(s) of the packet.
- *Rule Options* – which specify what content the rule should analyze, where it is located within the packet, the size of the packet, the message that is shown when the alert is triggered, and the rule ID.

3.1.3. Snort IPS

Snort IPS (Roesch, 2016) is the chosen tool to detect violations of the manufacturer's intentions and allow IoT device owners to block content communication according to their security and privacy preferences. Snort is an open-source rules based NIDPS (Network Intrusion Detection and Prevention Systems) software, to detect and prevent malicious attacks or information leakage, with the ability to analyze traffic in real time, log packets and generate alerts for users when packet content matches the rules that are defined.

To enable the option to block packets with content that violates the privacy and security of the owner from reaching their destination, it was necessary to configure Snort using the snort.conf file, selecting the DAQ (Data Acquisition Library) module Afpacket. This allows us to run Snort in inline mode with two network interfaces. Only in inline mode can Snort block the packets it captures; in normal mode it only generates alerts for the packets it has captured. Details are provided in *Chapter 4 – Tests and Evaluation*. Through the buffer_size_mb variable, we can define the memory to be assigned to the DAQ, considering the amount of traffic to be analyzed, the number of rules activated and the hardware Snort is running on. Additionally, it was also necessary to configure Snort to consider the rule files for each of the devices, both for the manufacturer's intents and for the IoT device owner's rules.

Figure 12 represents the command that allows Snort to run. It is made up of four main arguments and a fifth optional argument. The alerts mode (**1**)), your configuration file (**2**)), the interfaces where Snort "listens" for traffic to analyze (**3**)) and the operation mode (**4**)):



```
snortips@snortips:~$ sudo snort -A console -c /etc/snort/snort.conf -i ens33:ens34 -Q
```

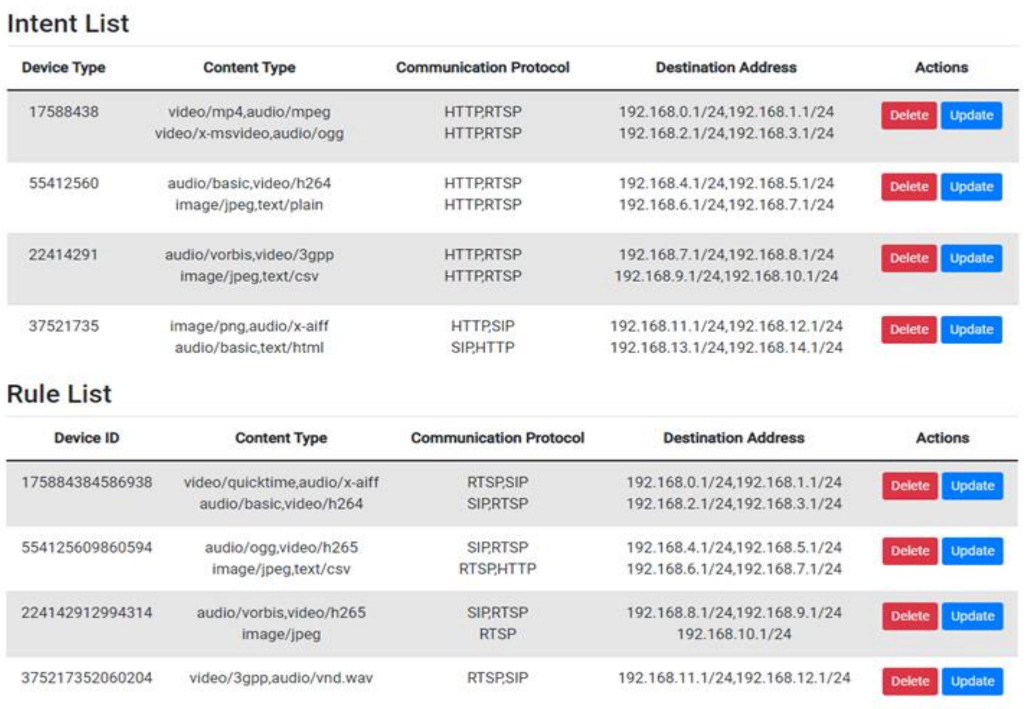
Figure 12 - Snort Launch Command

1) *- -A Console*: this argument configures Snort so that the alerts it generates are sent to the console.
2) *-c /etc/snort/snort.conf*: this argument allows you to choose the configuration file of the Snort settings that should be executed.
3) *-i ens33: ens34*: this argument chooses the interfaces to listen for traffic.
4) *-Q*: this argument allows you to run Snort in inline mode.

3.1.4. Web Management Interface

The Web Management Interface was designed to support the Intent & User Rule Management REST API and, in consequence, the Translator backend. This interface allows the device manufacturer to read, insert, update, and delete the device intents rules and enables the IoT device owner to read, insert, update, and delete the user-defined rules.

The intents and user-defined rules are displayed in tables in the intent list and rule list tabs, respectively, as shown in Figure 13. The intents table displays Device Type (TAC), Content Type, Communication Protocol, Destination Address and Actions (Delete and Update). The rules table displays Device ID (IMEI), Content Type, Communication Protocol, Destination Address and Actions (Delete and Update).


Intent List

Device Type	Content Type	Communication Protocol	Destination Address	Actions
17588438	video/mp4,audio/mpeg video/x-msvideo,audio/ogg	HTTP,RTSP HTTP,RTSP	192.168.0.1/24,192.168.1.1/24 192.168.2.1/24,192.168.3.1/24	Delete Update
55412560	audio/basic,video/h264 image/jpeg,text/plain	HTTP,RTSP HTTP,RTSP	192.168.4.1/24,192.168.5.1/24 192.168.6.1/24,192.168.7.1/24	Delete Update
22414291	audio/vorbis,video/3gpp image/jpeg,text/csv	HTTP,RTSP HTTP,RTSP	192.168.7.1/24,192.168.8.1/24 192.168.9.1/24,192.168.10.1/24	Delete Update
37521735	image/png,audio/x-aiff audio/basic,text/html	HTTP,SIP SIP,HTTP	192.168.11.1/24,192.168.12.1/24 192.168.13.1/24,192.168.14.1/24	Delete Update

Rule List

Device ID	Content Type	Communication Protocol	Destination Address	Actions
175884384586938	video/quicktime,audio/x-aiff audio/basic,video/h264	RTSP,SIP SIP,RTSP	192.168.0.1/24,192.168.1.1/24 192.168.2.1/24,192.168.3.1/24	Delete Update
554125609860594	audio/ogg,video/h265 image/jpeg,text/csv	SIP,RTSP RTSP,HTTP	192.168.4.1/24,192.168.5.1/24 192.168.6.1/24,192.168.7.1/24	Delete Update
224142912994314	audio/vorbis,video/h265 image/jpeg	SIP,RTSP RTSP	192.168.8.1/24,192.168.9.1/24 192.168.10.1/24	Delete Update
375217352060204	video/3gpp,audio/vnd.wav	RTSP,SIP	192.168.11.1/24,192.168.12.1/24	Delete Update

Figure 13 - Intents & Rules list table

To enter an intent the manufacturer must navigate to the intent form located in the intent tab, where he is greeted with the fields mentioned above. After he enters the data and saves the intent in the database, it is automatically translated by the system into a Snort rule, completely transparent to manufacturer. The translated intents will be written in a file that corresponds to the device TAC ID. When a device is deleted from the database by the manufacturer, the file that contains the corresponding Snort rules for that device is deleted as well. When the device intents are updated, the intents that existed previously are replaced with those entered by the manufacturer. The same process is followed for the user-defined rules entered by the IoT device owner in the rule tab, but with the slight difference that user-defined rules are written to a file that corresponds to the device IMEI ID. Figure 14 shows a representation of these web forms:

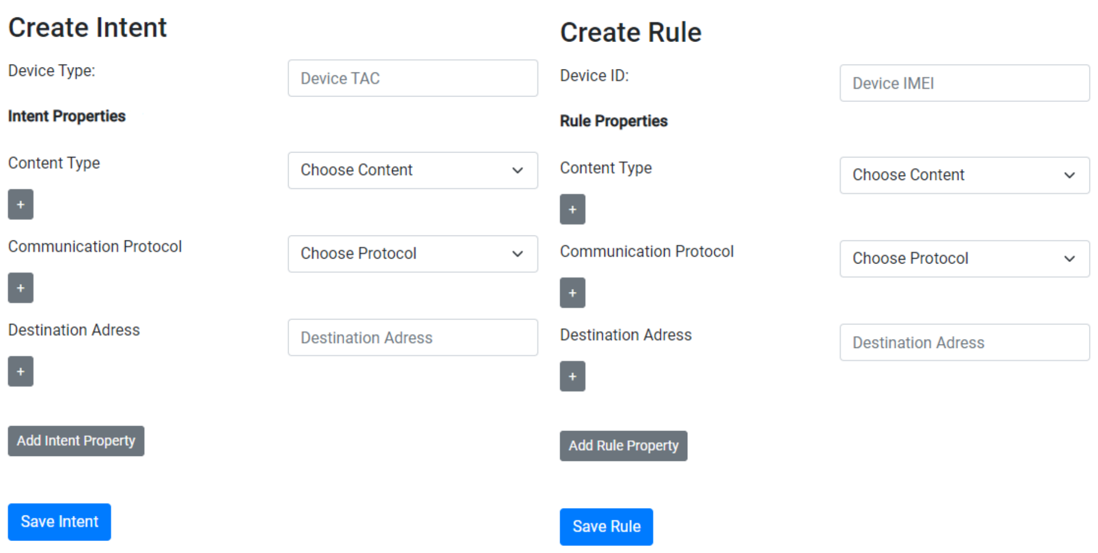
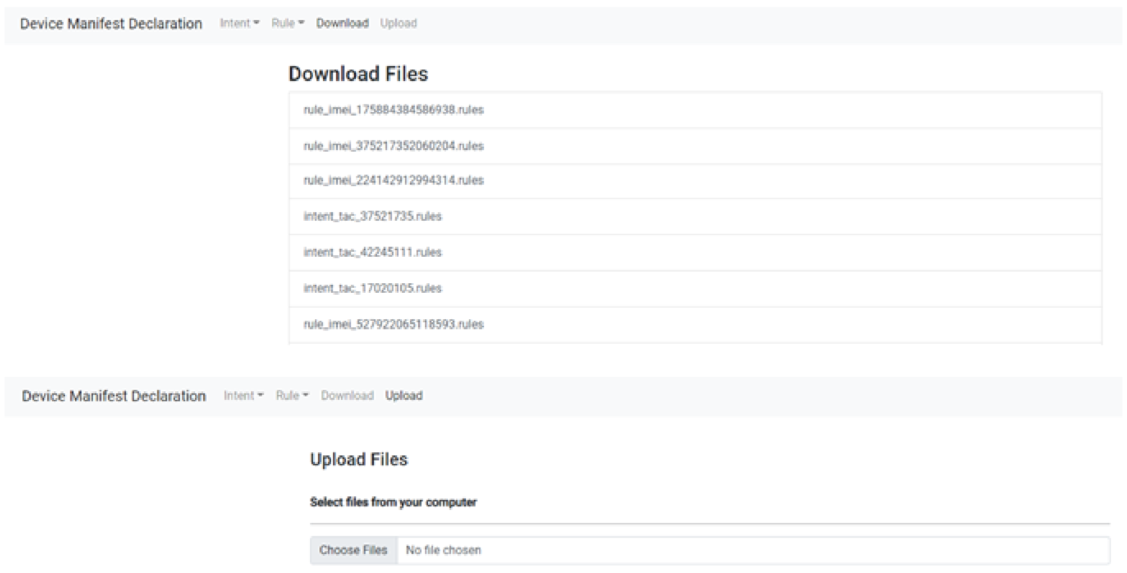


Figure 14 – Intent & Rule Insert/Update form

For more advanced users, who have experience of working with Snort and its rules, we have provided an appropriate interface, as shown in Figure 15, that allows them to download the text rule files to edit located in the download tab and upload them again located in the upload tab, replacing the existing ones with the ones they have modified to their liking.



Download Files

rule_imei_175884384586938.rules
rule_imei_375217352060204.rules
rule_imei_224142912994314.rules
intent_tac_37521735.rules
intent_tac_42245111.rules
intent_tac_17020105.rules
rule_imei_527922065118593.rules

Upload Files

Select files from your computer

Choose Files No file chosen

Figure 15 – Download/Update form

3.2. Snort Implementation Workflow

In this section, the Snort workflow is presented for each packet captured by the Network Traffic Filtering & Capturing service. We will be focusing on two different scenarios. The first is when the IoT device is violating the communication intents: this happens when the device deviates from its expected behavior as defined by the manufacturer. The second scenario is when the IoT device is violating the user-stipulated rules: this happens when the device deviates from the behavior the user sets for it.

The flow that represents the device intent communication violation scenario is presented in Figure 16. Snort starts by capturing a packet sent by an IoT device for inspection, so the decoder can analyze that packet to identify the transport protocol used and check for conflicts with the intent rules. When the decoder finishes the identification and classification process, it sends the packet to the pre-processor, which is responsible for preparing the packet to be processed more easily by the detection engine, which flags the traffic flow by looking for a match in the ruleset for traffic that is violating the defined policies. If a match is found, by TAC (device type) and content type in packet load, an intents violation alert is generated, followed by logging of the event. Otherwise, Snort takes another packet and repeats the same process. In this scenario, Snort acts in detection mode only, as it does not perform any action on the packet, but simply triggers an alert for a defined policy violation.

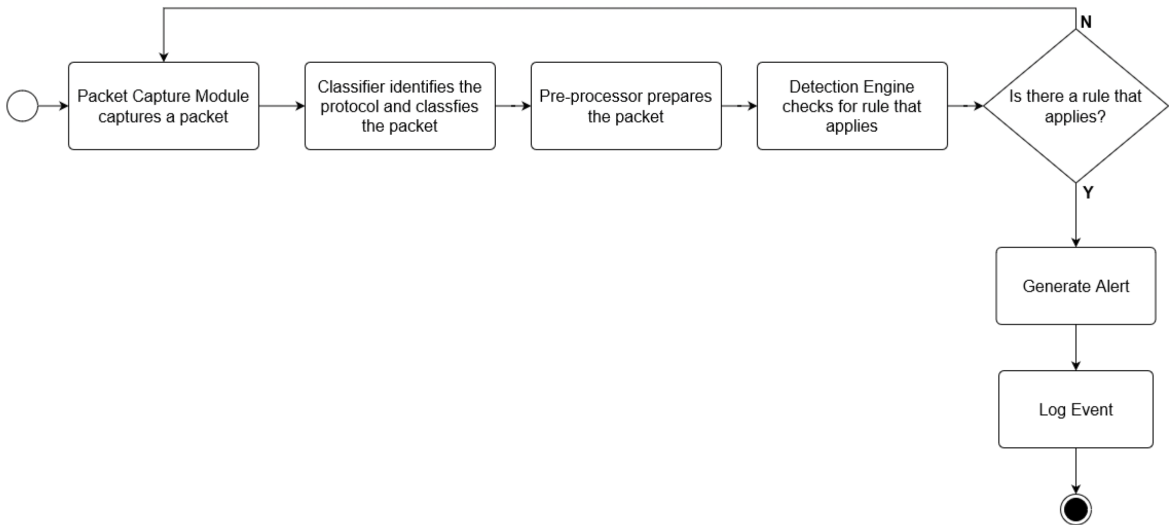

Packet Capture Module captures a packet
Classifier identifies the protocol and classfies the packet
Pre-processor prepares the packet
Detection Engine checks for rule that applies
Is there a rule that applies?
N
Y
Generate Alert
Log Event

Figure 16 – Intent scenario - System workflow

Considering the flow of the user-defined rules violation scenario presented in Figure 17, when it is detected that a device sends a packet that represents a violation of the privacy and security of its owner, the process is the same but, after generating the alert, Snort blocks and discards the packet that contains the device IMEI and the content type the IoT device owner has specified to be blocked, stopping it from reaching its destination, and then logs the event that occurred. Again, if no packet matches the user-defined rules, another is captured for analysis. In this scenario, Snort acts in prevention mode, because it stops the packet from reaching its destination by performing a drop action on the packet.

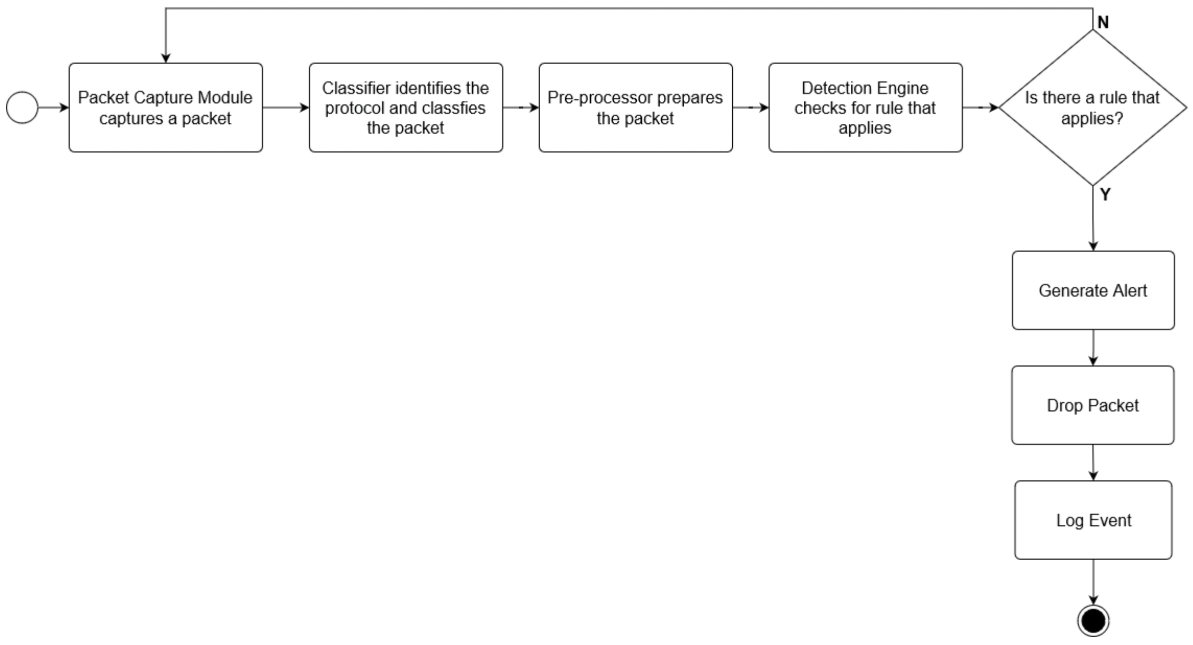

Packet Capture Module captures a packet
Classifier identifies the protocol and classfies the packet
Pre-processor prepares the packet
Detection Engine checks for rule that applies
Is there a rule that applies?
N
Y
Generate Alert
Drop Packet
Log Event

Figure 17 – Rule scenario - System workflow

When the manufacturer enters the intents for the IoT device using the Web Management Interface, the intents are stored and then sent from the Intent & User Management Rest API to the Translator REST API to be translated from an intent into alert Snort rules, because in the intent case our aim is only to alert the IoT device owner to the fact that their device is not complying with their communication intents, without blocking the content. These translated intents are written in the format and structure shown, following the Snort rule order defined in section *3.1.2. Translator (Integration layer with filtering tool)*. We will explain these Snort rules in greater detail in section *4.2. Functional Tests* of *Chapter 4 – Tests and Evaluation*:

- alert tcp $HOME_NET any -> [192.168.19.1/24,192.168.20.1/24] any (msg:"ALERT - DESTINATION ADDRESS NOT OK For Intent Of Device TAC 17020105. Intent{ContentType: [audio/mpeg, video/mp4] Destination Address: [192.168.19.1/24, 192.168.20.1/24]";flow:stateless; content:"|31 37 30 32 30 31 30 35|"; offset:4; depth:8; content:"|5b 61 75 64 69 6f 2f 6d 70 65 67 2c 76 69 64 65 6f 2f 6d 70 34 5d|"; distance:14; within:22;classtype:policy-violation; sid:1137585;)

When the IoT device owner enters the user-defined rules using the Web Management Interface, the rules are stored and then sent from the Intents & User Rule Management REST API to the Translator REST API to be translated from a user-defined rule into a drop Snort rule, because in the user rule case we want to prevent some of the content sent by the devices reaching the Internet and so violating owner privacy and security. These translated rules are written in the format and structure shown, following the Snort rule order defined in section *3.1.2. Translator (Integration layer with filtering tool)*. We will explain these Snort rules in greater detail in section *4.2. Functional Tests* of *Chapter 4 – Tests and Evaluation*:

- drop tcp $HOME_NET any -> [192.168.0.1/24,192.168.1.1/24] any (msg:"DROP – Rule Device IMEI 175884384586938{ Content Type: [video/quicktime, audio/x-aiff] Destination Address: [192.168.0.1/24, 192.168.1.1/24] }"; flow:stateless; content:"|31 37 35 38 38 34 33 38 34 35 38 36 39 33 38|"; offset:5; depth:15; content:"|5b 76 69 64 65 6f 2f 71 75 69 63 6b 74 69 6d 65 2c 61 75 64 69 6f 2f 78 2d 61 69 66 66 5d|"; distance:14; within:30;classtype:policy-violation; sid:8181915;)

In this chapter, we have described this project architecture and workflow in the most rigorous detail appropriate for this document. Even with the issues we faced during implementation of the architecture, we were able to implement the most important requirements to answer the research questions (Section *1.2. Research Question* of *Chapter 1 – Introduction*), as regards a mechanism to provide consumers with control over the information privacy and security of their IoT devices and achieve the research goals (Section *1.3. Research Goals* of *Chapter 1 – Introduction*) of developing a system prototype to perform traffic identification and filtering and violation reporting. Although we would have like to have been able to research and develop Truphone's existing solution based on PCRF and PCEF without compromising their quality of service, we are confident that they will be able to find a way to incorporate some of this research, based on Snort IPS, in their existing infrastructure, to provide a better service to their clients.

Chapter 4 – Tests and Evaluation

In this chapter, we describe the test environment implemented to perform the required tests (section *4.1. Test System Deployment*), the functional tests carried out (section *4.2. Functional Tests*) and the performance tests carried out (section *4.3. System Performance*).

4.1. Test System Deployment

The test deployment system presented in Figure 18 was created to allow us to test the violation of manufacturer-defined intents and violation of user-defined rules scenarios.

The intents scenario is important to test if the IoT device is not complying with the behavior defined by the manufacturer, because we need proof that guarantees to us those manufacturers and their devices can be trusted and are not being used to spy on innocent consumers. The user-defined rules scenario is important to test so that consumers possess a mechanism to protect themselves against an untrustworthy manufacturer who is using the devices in their home to gather information about them.

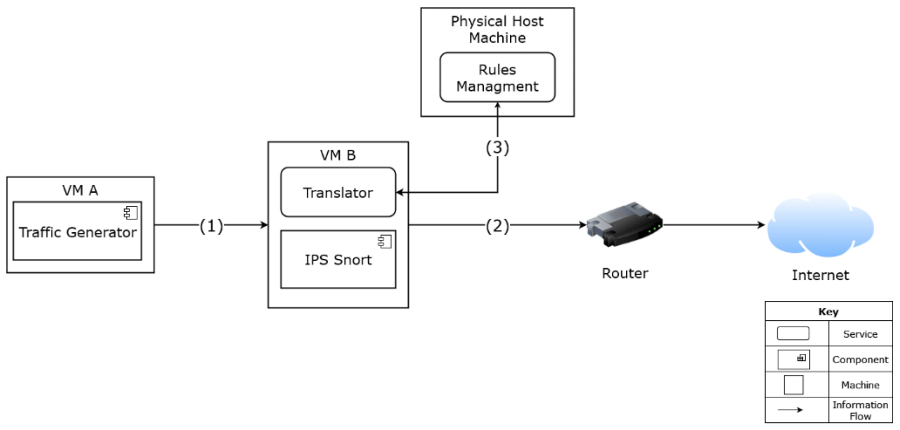


Figure 18 – Solution Testbed Architecture

This system consists of two different virtual machines, A and B, and a physical host machine. The system presented was inspired by the solution presented by (Hafeez, 2016) and described in section *2.2. Content Filtering & Blocking* of *Chapter 2 - Literature Review*. Our system, much like Hafeez's, uses Snort in intrusion prevention mode for network traffic control by making all the traffic from the Traffic Generator (VM A) pass through the Translator and Snort IPS (VM B) before reaching the Internet router allowing DPI analysis of the packets and application of the desired traffic control rules to the packet flow. In Table 1, we find the hardware and software specs of the system:

Table 1 - Testbed System Specs

Computer	
Hardware	
CPU	Core i7-10610U 1.8 GHZ
RAM Memory	32 GB
Graphics Card	Onboard Intel
Hard Disk	SSD 512 GB
Software	
OS	Windows 10 Pro
VM OS	Ubuntu 20.04.3 LTS
Hypervisor	Vmware Workstation Player 16.1.1
Ostinato Traffic Generator	1.0 (Free Open-Source Version)
WireShark	3.4.5

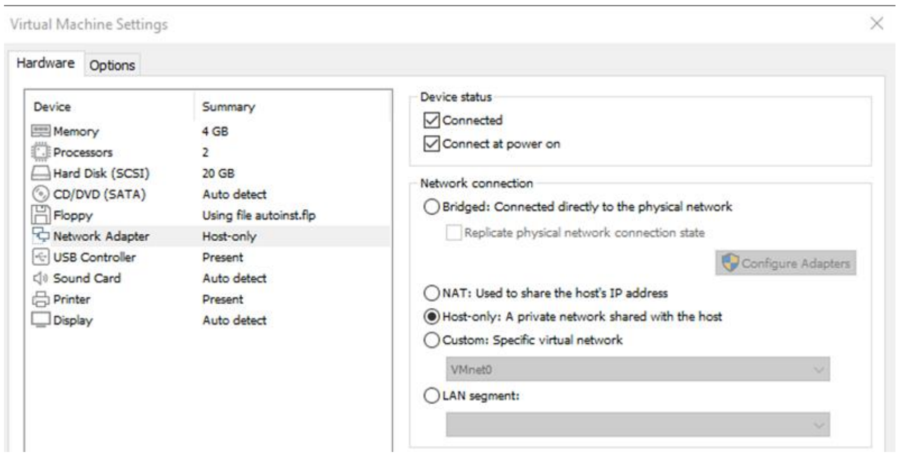
Virtual machine A was created with the purpose of representing an IoT device to generate network traffic to be collected and analyzed by Snort IPS, located on virtual machine B. (1) Figure 18.

Virtual machine B was created with the intended purpose of containing the traffic control and analysis software that collects and analyzes traffic from the IoT devices connected to this virtual machine. This machine acts as an internal server to check IoT device network traffic compliance with the traffic control rules and to forward it to the Internet if no violations occurred; if any violation did occur, an alert will be displayed, followed by blocking of the traffic. (2) Figure 18.

The physical host machine contains the Web Management Interface and the Intent & User Rule Management REST API that allow the manufacturers and IoT device users to specify intents and define rules for their devices by creating, reading, modifying and deleting rules, which are contained within a database. The Interface component of the Rules Management service calls the endpoint of the Translator service. (3) Figure 18.

The need for a virtual machine stems from the fact that the open-source software selected in this thesis is designed to run on open-source Linux core-based operating systems. Also, as this is open-source software there are no additional expenses for acquiring the software and its licenses, nor any need for extra physical hardware that can be perfectly replicated using appropriate open-source virtualization software with only one optimal physical host machine.

VM A, as shown in Figure 19, is configured to use only one network adapter. The adapter is set to Host Only and is used for the connection between virtual machines A and B, representing the network gateway of VM A.


Virtual Machine Settings

Hardware | Options

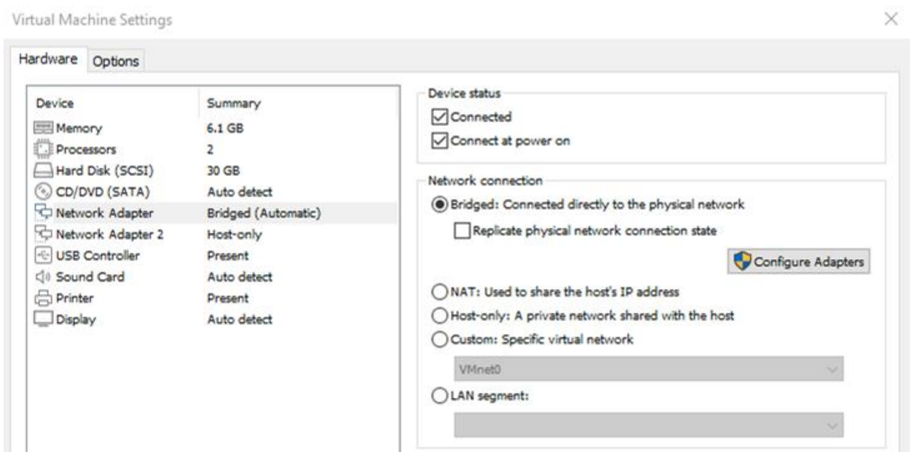
Device	Summary
Memory	4 GB
Processors	2
Hard Disk (SCSI)	20 GB
CD/DVD (SATA)	Auto detect
Floppy	Using file autoinst.flp
Network Adapter	Host-only
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

Device status
[✓ Connected]
[✓ Connect at power on]

Network connection
Bridged: Connected directly to the physical network
Replicate physical network connection state
Configure Adapters
NAT: Used to share the host's IP address
[✓ Host-only: A private network shared with the host]
Custom: Specific virtual network
VMnet0
LAN segment:

Figure 19 -Traffic Generator (VM A) network adapter

VM B, as shown in Figure 20, is configured to use two network adapters. Adapter 1 is configured as Bridged: it is used to connect the virtual machine to the main network, where the router is located. Adapter 2 is configured as Host Only: it is used to connect VM B to VM A, acting as gateway to the VM A network.


Virtual Machine Settings

Hardware | Options

Device	Summary
Memory	6.1 GB
Processors	2
Hard Disk (SCSI)	30 GB
CD/DVD (SATA)	Auto detect
Network Adapter	Bridged (Automatic)
Network Adapter 2	Host-only
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

Device status
[✓ Connected]
[✓ Connect at power on]

Network connection
[✓ Bridged: Connected directly to the physical network]
Replicate physical network connection state
Configure Adapters
NAT: Used to share the host's IP address
Host-only: A private network shared with the host
Custom: Specific virtual network
VMnet0
LAN segment:

Figure 20 – Translator and Snort IPS (VM B) network adapters

A open source version of the Ostinato(P, 2010) software was chosen to generate traffic streams, so that we can simulate the various devices that are present on an IoT network. Ostinato is an open-source tool that allows us to test bandwidth, create and configure multiple streams, specify how data is sent, set the number of packets or bursts to be sent per second, view real-time statistical data for network monitoring and measurement, read statistical measurements of lost packets per stream, and offers support for the most common standard protocols.

Using this software, we created multiple streams to cover the IoT device owner rules and the manufacturer-specified intents scenarios. For these streams, we defined the protocol data settings, the source and destination IP address of the packet, the transport protocol and the HTTP, RTSP or SIP text messages. In the intent case, the TAC of the device and the content it transmits and, in the case of the IoT device owner's rule, the IMEI of the device and the content it transmits.

Finally, we defined the stream control settings, the type we wish to send, whether packet or burst, the number of packets or bursts, the rate of packets or bursts per second, and what action is to be taken when the stream ends, as explained above.

4.2. Functional Tests

In this section, we introduce the functional scenarios used in the proposal: one that tests intent violations and another that tests the user's control over their devices’ communication. We also present and discuss the test results.

These tests are important to ensure that the system we have implemented performs as expected. To prove this, we must test the system to see if it is able to detect violations of the declared manufacturer intents and generate alerts, whenever the IoT device transmits packets that carry information that does not match the specifications of the intents manifest of the IoT device manufacturer. To conduct this test, the system creates test packets using the traffic generator to simulate misbehavior of the IoT device. These packets must contain information that diverges from the information specified in the manufacturer's intents manifest. This test is successful if alerts are triggered for each packet that violates the specifications of the manufacturer's intents manifest. Thus such devices will not be able to compromise the privacy and security of the owner, who can be assured that devices do what is stated in their manifest.

To prove that the IoT device owner has control over the privacy and security of his device's communication, we must create test packets using the same traffic generator as mentioned above with content that violates user-defined rules, so when the system detects this content, it blocks it from reaching its destination. In this manner, the system implemented can assure the IoT device owner of control over their device's communication and guarantee their privacy and security.

4.2.1. Description of Test Scenario I – Violations of the manifest of intents

In this test scenario, we simulate network traffic that violates manufacturers' intents and analyze the system’s reaction to it. To do so, we use the intents of three different devices, which have been specified by the manufacturer, as set out in Table 2:

Table 2 - Test Intents

Device	Device Type (TAC)	Declared Content-Type	Declared Communication Protocol	Declared Destination Addresses
Samsung Android Tablet	09951213	[video/mp4, audio/mpeg] [video/x-msvideo, audio/ogg]	[HTTP, RTSP] [HTTP, RTSP]	[192.168.21.0/24,192.168.22.0/24] [192.168.23.0/24,192.168.24.0/24]
Apple IOS Smartphone	42245111	[audio/basic, video/h264] [image/jpeg, text/plain]	[HTTP, RTSP] [HTTP, RTSP]	[192.168.25.0/24,192.168.26.0/24] [192.168.27.0/24,192.168.28.0/24]
Huawei Android Smartphone	33292418	[audio/mpeg, video/mp4]	[SIP, RTSP]	[192.168.29.0/24,192.168.30.0/24]

Analyzing the intents defined by the manufacturers in the previous table, we see how they have specified communication intents for three well-known IoT devices. The unique eight-digit TAC is required to identify a device by its brand and model on the network that it is connected to. The content type that device sends to the Internet is specified using MIME (Multipurpose Internet Mail Extensions) types. This content can originate from the device camera, microphone, location sensor, humidity sensor or other device sensors. The communication protocol represents the protocol that is used to transport the TAC and content-type information to one of the IPs defined in the destination address.

The intents content-type and destination addresses, as shown in Table 2, must be denied so that it is possible to validate whether the manufacturer-defined intent is fulfilling the device communication intentions. This way, we can evaluate if the content type and destination addresses defined in the device intents is the same as the content type and destination address that the device sends to the Internet. If this is true, the user should not be confronted with them in an execution environment, as they comply with the manufacturer-defined intent. If not, the user should be presented with alert messages in an execution environment, alerting them to non-compliance with the manufacturer-defined intent.

After the intents are created and translated, it is necessary to create network traffic, meaning the packets, content type and destination address, which must be different from what is specified by the manufacturer, so that the alerts can be triggered. Table 3 shows an example of a packet whose content type does not match the content type specified by the manufacturer, but whose destination address corresponds to one of the ranges. Table 4 shows a packet with a destination address that differs from the one specified, but whose contents match those specified.

Table 3 - Content test intent packet

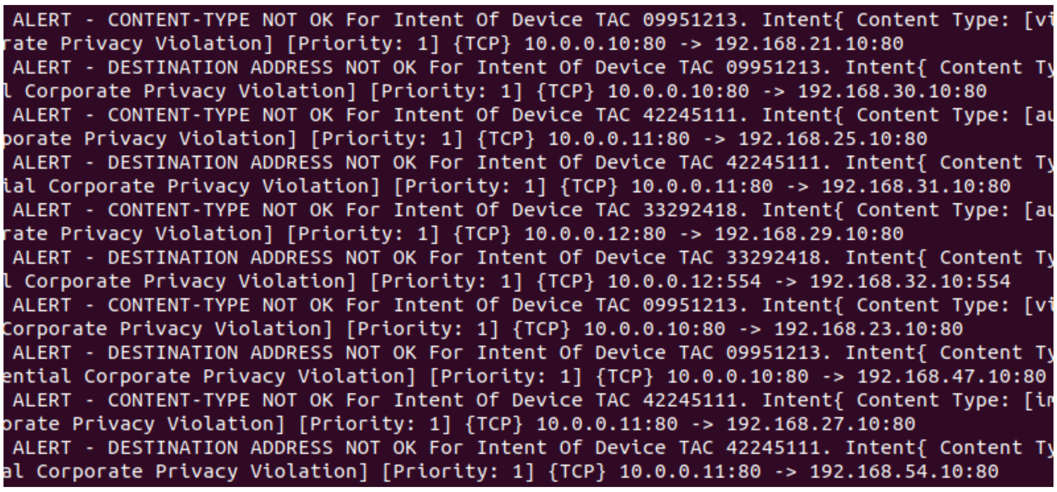
Protocol Data	
Source IP: 10.0.0.10	Destination IP: 192.168.21.10
Text Protocol	
TAC:09951213 Content-Type: [text/plain, video/h264]	

Table 4 - Destination test intent packet

Protocol Data	
Source IP: 10.0.0.10	Destination IP: 192.168.30.10
Text Protocol	
TAC:09951213 Content-Type: [video/mp4, audio/mpeg]	

4.2.2. Test Scenario I - Results Presentation

Figure 21 represents the Snort messages displayed when it is detected that the content and destination address of the previously created packets do not match the content and destination address specified by the manufacturer in the intents.



```
 ALERT - CONTENT-TYPE NOT OK For Intent Of Device TAC 09951213. Intent{ Content Type: [vi
rate Privacy Violation] [Priority: 1] {TCP} 10.0.0.10:80 -> 192.168.21.10:80
 ALERT - DESTINATION ADDRESS NOT OK For Intent Of Device TAC 09951213. Intent{ Content Ty
l Corporate Privacy Violation] [Priority: 1] {TCP} 10.0.0.10:80 -> 192.168.30.10:80
 ALERT - CONTENT-TYPE NOT OK For Intent Of Device TAC 42245111. Intent{ Content Type: [au
porate Privacy Violation] [Priority: 1] {TCP} 10.0.0.11:80 -> 192.168.25.10:80
 ALERT - DESTINATION ADDRESS NOT OK For Intent Of Device TAC 42245111. Intent{ Content Ty
ial Corporate Privacy Violation] [Priority: 1] {TCP} 10.0.0.11:80 -> 192.168.31.10:80
 ALERT - CONTENT-TYPE NOT OK For Intent Of Device TAC 33292418. Intent{ Content Type: [au
rate Privacy Violation] [Priority: 1] {TCP} 10.0.0.12:80 -> 192.168.29.10:80
 ALERT - DESTINATION ADDRESS NOT OK For Intent Of Device TAC 33292418. Intent{ Content Ty
l Corporate Privacy Violation] [Priority: 1] {TCP} 10.0.0.12:554 -> 192.168.32.10:554
 ALERT - CONTENT-TYPE NOT OK For Intent Of Device TAC 09951213. Intent{ Content Type: [vi
Corporate Privacy Violation] [Priority: 1] {TCP} 10.0.0.10:80 -> 192.168.23.10:80
 ALERT - DESTINATION ADDRESS NOT OK For Intent Of Device TAC 09951213. Intent{ Content Ty
ential Corporate Privacy Violation] [Priority: 1] {TCP} 10.0.0.10:80 -> 192.168.47.10:80
 ALERT - CONTENT-TYPE NOT OK For Intent Of Device TAC 42245111. Intent{ Content Type: [in
orate Privacy Violation] [Priority: 1] {TCP} 10.0.0.11:80 -> 192.168.27.10:80
 ALERT - DESTINATION ADDRESS NOT OK For Intent Of Device TAC 42245111. Intent{ Content Ty
al Corporate Privacy Violation] [Priority: 1] {TCP} 10.0.0.11:80 -> 192.168.54.10:80
```

Figure 21 - Intent violation alerts

Analyzing the messages displayed, we see that Snort tell us that, for example, the device with the *TAC 099551213* has committed four intent violations. The first was that the content type the device was sending didn't match the content type the manufacturer specified in their intent: the manufacturer specified the device was supposed to send the content *[video/mp4, audio/mpeg]*, but instead, after analyzing the packet on Wireshark from *source IP 10.0.0.10:80* to *destination IP 192.168.21.10:80* and *TAC 099551213*, we saw that the content the device actually sent was *[text/plain, video/h264]*. The second was that the destination address to which the packet was intended to go didn't match the address the manufacturer specified in their intent: the manufacturer specified that the device was supposed to send the content *[video/mp4, audio/mpeg]* to one IP address from the following IP ranges *[192.168.21.1/24, 192.168.22.1/24]*, but instead, after analyzing the packet from *source IP 10.0.0.10:80* and *TAC 099551213*, we saw that the actual *destination IP address* was *192.168.30.10*. The two other intent violations are identical to those described above only with different destination addresses and content types.

In the Snort log folder, we can view the files, with the timestamp of capture and where the captured packets are recorded, which can be opened in a program such as Wireshark or another relevant program. These logs represent the packets whose content triggered the alarms for intent manifest violations. Thus we possess detailed reports that users can check for the number of traffic control rule violations and the number of alarms that were triggered in the process. Depending on how Snort alarm mode is configured, we also can access the log files in the same folder of the Snort events to check the number of alarms and rules that were triggered.

4.2.3. Test Scenario I - Discussion of Results

In this test, we have shown that the detection and alarm functional process succeeded. The system was able to detect that the virtual device content and destination address present in the test packets was not compliant with the content type and destination address specified in the device intentions. It therefore generated a log of alerts for these events, reporting that content or content destination was not correct, according to the device intent manifest specifications. If the device was indeed complying with the device intents, no log of alerts would have been generated, meaning no violations of intents occurred that could have jeopardized the privacy and security of the IoT device owner.

In the eventuality of a manufacturer failing to correctly specify their device's real communication intentions, our system will be able to inform the IoT device owner whether their device is reliable or not, meaning it will tell the device owner if the device is complying with the manufacturer intents or not.

4.2.4. Description of Test Scenario II - Violations of user-defined rules

In this test scenario, we simulate violation of defined rules of privacy and information security by three devices and how the system is supposed to react when such an event occurs. To do this, we use the following rules specified by the owner, which enable the submission of unapproved content to be blocked, seen in Table 5:

Table 5 - Test Rules

Device	Device ID (IMEI)	Content Type allowed by device owner	Communication Protocol allowed by device owner	Destination Address allowed by device owner
Samsung Android Tablet	099512133857038	[video/quicktime, audio/x-aiff]	[RTSP, SIP]	[192.168.21.0/24,192.168.22.0/24]
		[audio/basic, video/h264]	[SIP, RTSP]	[192.168.23.0/24,192.168.24.0/24]
Apple IOS Smartphone	422451118095078	[audio/ogg, video/h265]	[SIP, RTSP]	[192.168.25.0/24,192.168.26.0/24]
		[image/jpeg, text/csv]	[RTSP, SIP]	[192.168.27.0/24,192.168.28.0/24]
Huawei Android Smartphone	332924181938544	[image/jpeg]	[RTSP]	[192.168.29.0/24]

In the table above we show user-defined rules to prevent certain types of content from three IoT devices leaking to the Internet, resulting in possible violations of users' privacy and security. The user rules seen here are defined by their IMEI, which acts as the device identifier, and the content type that the user has chosen to block, also specified using the MIME types.

The user-defined rules, shown in Table 5, are written to prevent packets that meet the specified conditions reaching the Internet. Whenever it is found that a device with a particular IMEI is submitting the specified content to one of the specified addresses, the rule should be activated and the event logged, followed by immediate rejection of the packet.

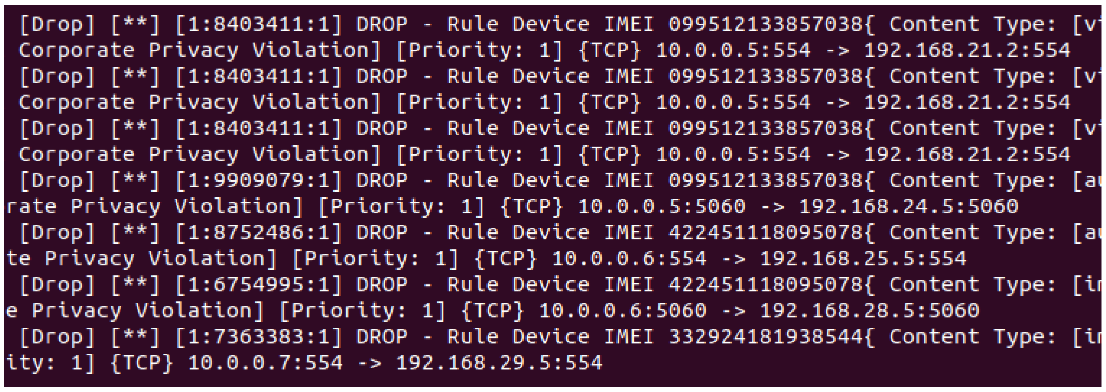
After creating the rules and their translation, it is necessary to create the packets. The content must correspond to what is specified by the owner, so that the packet can be rejected for being in violation of the rules defined by the owner. Table 6 shows an example of a packet whose contents are specified in the rule to be dropped.

Table 6 - Content test rule packet

Protocol Data	
Source IP: 10.0.0.5	Destination IP: 192.168.21.2
Text Protocol	
IMEI: 099512133857038 Content-Type: [text/plain, video/h264]	

4.2.5. Test Scenario II – Results Presentation

Figure 22 represents the Snort messages that inform us that a packet has been discarded for being in violation of the user-defined rules.



```
 [Drop] [**] [1:8403411:1] DROP - Rule Device IMEI 099512133857038{ Content Type: [v
 Corporate Privacy Violation] [Priority: 1] {TCP} 10.0.0.5:554 -> 192.168.21.2:554
 [Drop] [**] [1:8403411:1] DROP - Rule Device IMEI 099512133857038{ Content Type: [v
 Corporate Privacy Violation] [Priority: 1] {TCP} 10.0.0.5:554 -> 192.168.21.2:554
 [Drop] [**] [1:8403411:1] DROP - Rule Device IMEI 099512133857038{ Content Type: [v
 Corporate Privacy Violation] [Priority: 1] {TCP} 10.0.0.5:554 -> 192.168.21.2:554
 [Drop] [**] [1:9909079:1] DROP - Rule Device IMEI 099512133857038{ Content Type: [a
rate Privacy Violation] [Priority: 1] {TCP} 10.0.0.5:5060 -> 192.168.24.5:5060
 [Drop] [**] [1:8752486:1] DROP - Rule Device IMEI 422451118095078{ Content Type: [a
te Privacy Violation] [Priority: 1] {TCP} 10.0.0.6:554 -> 192.168.25.5:554
 [Drop] [**] [1:6754995:1] DROP - Rule Device IMEI 422451118095078{ Content Type: [i
e Privacy Violation] [Priority: 1] {TCP} 10.0.0.6:5060 -> 192.168.28.5:5060
 [Drop] [**] [1:7363383:1] DROP - Rule Device IMEI 332924181938544{ Content Type: [i
ity: 1] {TCP} 10.0.0.7:554 -> 192.168.29.5:554
```

Figure 22 - Rule drop packet alert

Analyzing the messages displayed, we see that Snort informs us that, for example, the device with the *IMEI 422451118095078* has committed two user-defined policy violations. The first was that the *content type* of the packet sent by the device, viewed in Wireshark for the packet sent from *source IP address 10.0.0.5* to *destination IP address 192.168.25.5*, matches the *content type* that the user has defined to be blocked *[audio/ogg, video/h265]*. The second was that the content type of the packet sent by the device, viewed in Wireshark for the packet sent from *source IP address 10.0.0.5* to *destination address IP 192.168.28.5*, matches the *content type* that the user has defined to be blocked *[image/jpeg, text/csv]*.

In the same location as mentioned in the previous subsection, we can also see the packet files that contain the captured packets that have violated the user-defined rules, with the information about the content they were carrying and the capture timestamp. The log files that contain the event logs of the rules that were triggered by the test packets are also located in the same folder.

4.2.6. Test Scenario II - Discussion of Results

In this test, we have shown that the detection and blocking process succeeded. The system was able to detect that the virtual IoT device content present in the test packets was in violation of the user-defined rules, resulting in compromise of the user-defined privacy and security rules. It therefore dropped the packets that violated the user-defined rules, stopping them from reaching their destination.

The results of this test prove that the traffic control rules defined by the user enable them to control how their device communicates, ensuring that the user can guarantee the privacy and security of their personal information when the intentions of manufacturers and the actions of their devices are not to be trusted.

4.3. System Performance

In this section, we describe the performance tests that were performed for 1000, 5000, and 10000 packets generated by the selected traffic generator for different packet throughputs. For this test, we used 10 devices for which 19 intent rules and 15 user-defined rules were created. The variables selected for the results were:

- *Packets Received* – This variable represents the number of packets that were captured for inspection.
- *Packets Analyzed* – This variable represents the number of packets that were parsed from packets that were received.
- *Dropped Packets* – This variable represents the number of packets that were not found and therefore not analyzed by the analysis component.
- *Packets Whitelisted/Blacklisted* – This variable represents the number of packets whose content and destination address were in violation of manufacturer's specifications or IoT device owner rules.
- *Analysis Runtime* – This variable represents the time taken to perform analysis and classification of packets received by the traffic capture component.
- *Dump Runtime* - This variable represents the time taken to generate and send packets to the network.
- *Delay* - This variable represents the additional expense required to perform analysis of the packets generated.

The aim of these performance tests is to test how Snort performs in the worst-case scenarios, to measure if high throughputs have a major impact on the time taken to analyze the content of the packet and detect that content is in violation of manufacturer intents or user-defined rules.

4.3.1. Intents Scenario – Results Presentation

Figure 23 shows the data collected from the tests performed for 1000 packets, these are transmitted in function of time per second. In this graph, we display the time it takes Snort to analyze the required number of packets (Analysis Runtime), the time it takes the Ostinato traffic generator to generate the required number of packets and send them to the network (Dump Runtime), and finally the delay between the Snort analysis of the captured packets and the generation of packets by Ostinato (Delay).

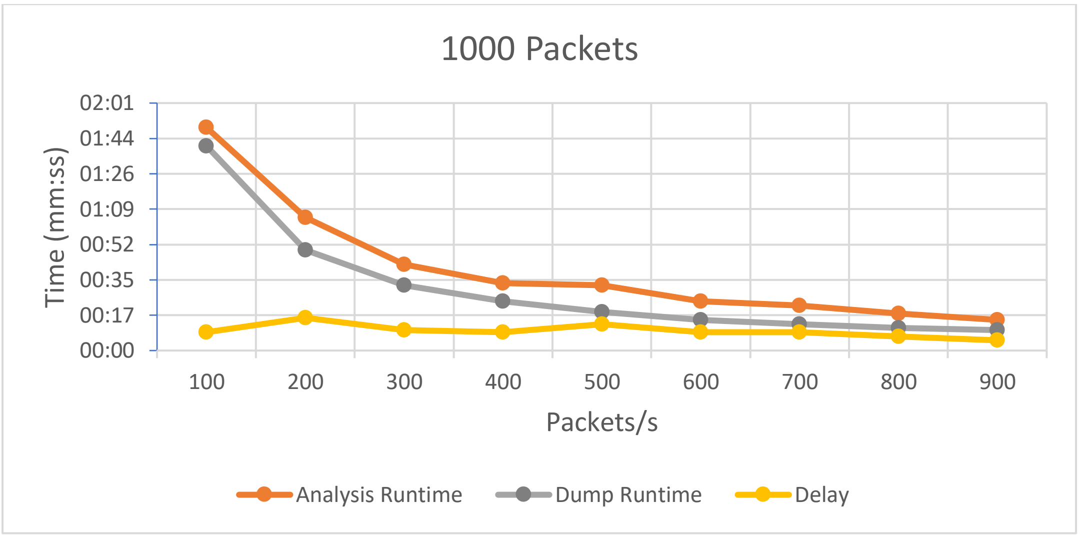


Figure 23 - 1000 Packets runtime data

The highest Snort analysis runtime registered for this test is *1 minute and 5 seconds* for throughput of 200 packets per second and the lowest is *15 seconds* for throughput of 900 packets per second. The average Snort analysis runtime of this test is approximately *40 seconds*. The highest packet dump runtime registered is *1 minute and 40 seconds* for throughput of 100 packets/s and the lowest is *10 seconds* for throughput of 900 packets/s. The average packet dump runtime in this test is approximately *30 seconds*. The longest delay we recorded was *13 seconds* for throughput of 500 packets/s and the lowest was *5 seconds* for throughput of 900 packets/s. The average delay in this test was approximately *10 seconds*.

Table 7 shows the remaining data collected for other variables. All the packets captured by Snort during the execution of this test were analyzed, meaning no packets were skipped or dropped during that time. Of all the packets captured, a percentage between 3% and 14% corresponded to packets whose content was in violation of the manufacturer's intents on the device manifest, while the remaining percentage corresponded to miscellaneous traffic capture by Snort.

Table 7 - 1000 packets variable data

Packets/s Throughput	**Packets Captured**	**Packets Analyzed**	**Packets Dropped**	**Packets Whitelisted**
100	11311	100%	0%	88,39%
200	10837	100%	0%	92,27%
300	11482	100%	0%	86,83%
400	10436	100%	0%	95,80%
500	10260	100%	0%	97,45%
600	10272	100%	0%	97,31%
700	10188	100%	0%	98,13%
800	51632	100%	0%	96,82%
900	51432	100%	0%	97,21%

Figure 24 shows the data collected from the tests performed for 5000 packets. The longest Snort analysis runtime recorded for this test was *8 minutes and 36 seconds* for throughput of 100 packets/s and the shortest was *1 minute and 3 seconds* for throughput of 900 packets/s. The average Snort analysis runtime was approximately *2 minutes and 49 seconds*. The longest packet dump runtime recorded in this test was *8 minutes and 18 seconds* for throughput of 100 packets/s and the shortest was *11 seconds* for throughput of 4000 packets/s. The longest packet dump runtime was approximately *2 minutes and 12 seconds*. The longest delay recorded in this test was *1 minute and 24 seconds* for throughput of 500 packets/s and the shortest was *8 seconds* for throughput of 900 packets/s. The average delay was approximately *37 seconds*.

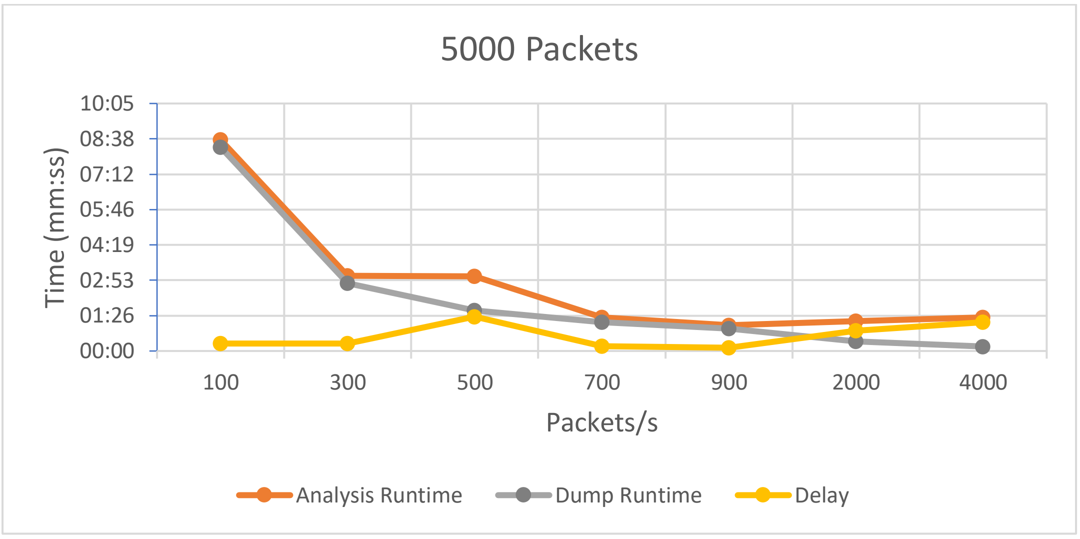


Figure 24 - 5000 Packets runtime data

Table 8 shows we can view the remaining data collected for other variables. All the packets captured by Snort during the execution of this test were analyzed, meaning no packets were skipped or dropped during that time. Of all the packets captured, a percentage between 2% and 31% corresponded to packets whose content was in violation of the manufacturer's intents on the device manifest, while the remaining percentage corresponded to miscellaneous traffic capture by Snort.

Table 8 - 5000 Packets variable data

Packets/s Throughput	**Packets Captured**	**Packets Analyzed**	**Packets Dropped**	**Packets Whitelisted**
100	71695	100%	0%	69,72%
300	54221	100%	0%	92,21%
500	65098	100%	0%	76,69%
700	51253	100%	0%	97,52%
900	50564	100%	0%	98,85%
2000	51632	100%	0%	96,82%
4000	51432	100%	0%	97,21%

In Figure 25, we can view the data collected from the tests performed for 10000 packets. The longest Snort analysis runtime recorded for this test was *16 minutes and 45 seconds* for throughput of 100 packets/s and the shortest was *1 minute and 36 seconds* for throughput of 9000 packets/s. The average Snort analysis runtime was approximately *4 minutes and 23 seconds*. The longest packet dump runtime recorded in this test was *16 minutes and 39 seconds* for throughput of 100 packets/s and the shortest was *10 seconds* for throughput of 9000 packets/s. The average packet dump runtime was approximately *3 minutes and 17 seconds*. The longest delay recorded in this test was *3 minutes and 15 seconds* for throughput of 9000 packets/s and the shortest was *6 seconds* for throughput of 100 packets/s. The average delay was approximately *1 minute and 37 seconds*.

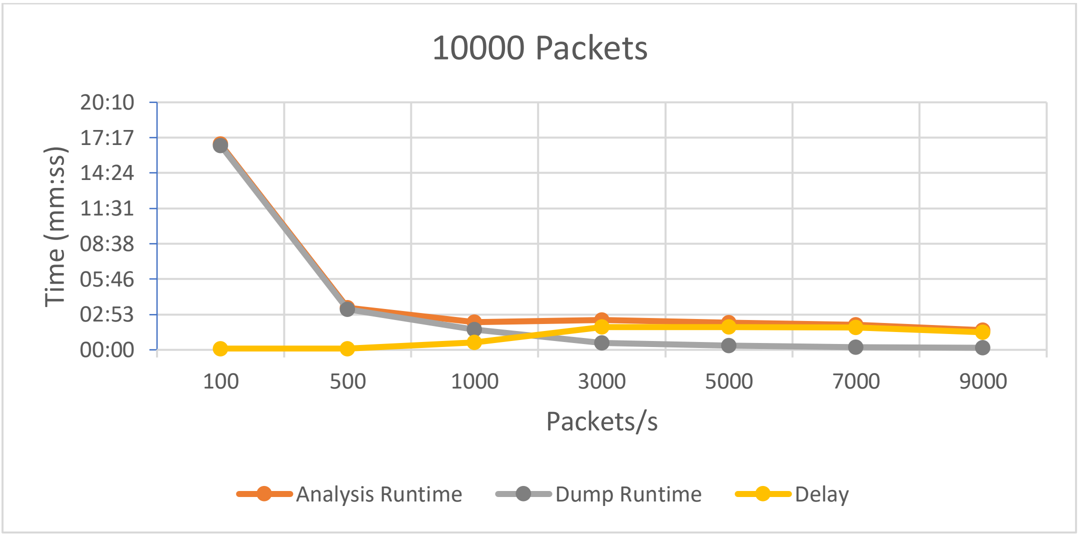


Figure 25 - 10000 Packets runtime data

Table 9 shows the remaining data collected for other variables. All the packets captured by Snort during the execution of this test were analyzed, meaning no packets were skipped or dropped during that time. Of all the packets captured, a percentage between 2% and 12% corresponded to packets whose content was in violation of the manufacturer’s intents on the device manifest, while the remaining percentage corresponded to miscellaneous traffic capture by Snort.

Table 9 - 10000 packets variable data

Packets/s Throughput	Packets Captured	Packets Analyzed	Packets Dropped	Packets Whitelisted
100	113019	100%	0%	88,48%
500	102240	100%	0%	97,62%
1000	101391	100%	0%	98,62%
3000	105846	100%	0%	94,25%
5000	100941	100%	0%	98,71%
7000	102817	100%	0%	94,81%
9000	101165	100%	0%	96,37%

4.3.2. Intent Scenario – Discussion of Results

In the results displayed in subsection *4.3.1. Intents Scenario – Results Presentation*, we were able to prove that the system implemented was able to detect and handle multiple packets at various throughputs, without ever showing service breaks during the analysis for high packet loads, maintaining a cadence of approximately *13 seconds* per packet without missing or skipping packets.

According to the results shown in Table 10, analysis of the captured packets took longest in the 5000 and 10000 packets per second tests when the rate of packet throughput was the lowest at 100 packets per second, because Snort could analyze only 100 packets at a time. With increased throughput, we saw that time decrease progressively; as we came closer to reaching rates of 1000, 5000 and 10000 packets per second the difference between the different rates had decreased approximately *2 seconds* for the 1000-packet test and *22 seconds* for the 5000 and 10000-packet tests.

With that time decrease, the analysis displayed a tendency to become more consistent, which meant that the system started to stabilize at around *500 packets/s* in the 1000-packet test, *700 packets/s* in the 5000-packet test and *1000 packets/s* in the 10000-packet test. This gives a packet analysis average after consistency of approximately 4 *seconds* in the 1000-packet test, *10 seconds* in the 5000-packet test and *19 seconds* in the 10000-packet test. This meant that the test system did not possess the capacity of improving is performance further of what already had achieved at that point.

Table 10 - Intent Scenario Results Summary

	1000 Packets/s	5000 Packets/s	10000 Packets/s
Analysis Runtime High/Low	1m49s/15s	8m36s /1m03s	16m45s/ 1m36s
Dump Runtime High/Low	1m40s/10s	8m18s /11s	16m39s / 10s
Delay Runtime High/Low	16s /5s	1m24s / 8s	1m48s / 6s
Average Analysis Runtime	40s	2m49s	4m23s
Average Dump Runtime	30s	2m12s	3m17s
Average Delay Runtime	10s	37s	1m06s

4.3.3. Rules Scenario – Results Presentation

Figure 26 shows the data collected from the tests performed for 1000 packets, these are transmitted in function of time per second. In this graph, we display the time it takes Snort to analyze the required number of packets (Analysis Runtime), the time it takes the Ostinato traffic generator to generate the required number of packets and send them to the network (Dump Runtime), and finally the delay between the Snort analysis of the captured packets and the generation of packets by Ostinato (Delay).

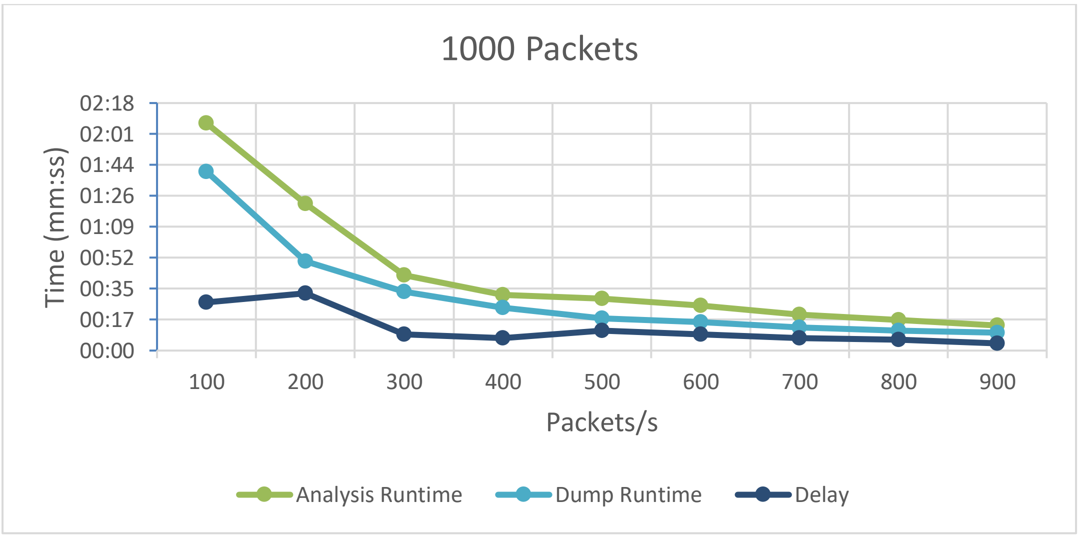


Figure 26 - 1000 Packets runtime data

The longest Snort analysis runtime recorded in this test was *2 minutes and 7 seconds* for throughput of 100 packets per second and the shortest was *14 seconds* for throughput of 900 packets per second. The average Snort analysis runtime in this test was approximately *43 seconds*. The longest packet dump runtime recorded was *1 minute and 40 seconds* for throughput of 100 packets/s and the shortest was *10 seconds* for throughput of 900 packets/s. The average packet dump runtime in this test was approximately *31 seconds*. The longest delay we recorded was *32 seconds* for throughput of 200 packets/s and the shortest was *4 seconds* for throughput of 900 packets/s. The average delay in this test was approximately *12 seconds*.

Table 11 shows the remaining data collected for other variables. All the packets captured by Snort during the execution of this test were analyzed, meaning no packets were skipped or dropped during that time. Of all the packets captured, a percentage between 79% and 98% corresponded to packets whose content was in violation of the user-defined rules, while the remaining percentage corresponded to miscellaneous traffic capture by Snort.

Table 11 - 1000 Packets variable data

Packets/s Throughput	**Packets Captured**	**Packets Analyzed**	**Packets Dropped**	**Packets Blacklisted**
100	11936	100%	0%	79,58%
200	12035	100%	0%	82,63%
300	10407	100%	0%	96,08%
400	10581	100%	0%	94,48%
500	10252	100%	0%	97,51%
600	10298	100%	0%	97,10%
700	10191	100%	0%	98,12%
800	11081	100%	0%	90,24%
900	10175	100%	0%	98,26%

Figure 27 shows the data collected from the tests performed for 5000 packets. The longest Snort analysis runtime recorded in this test was *9 minutes and 23 seconds* for throughput of 100 packets per second and the shortest was *1 minute and 9 seconds* for throughput of 4000 packets per second. The average Snort analysis runtime in this test was approximately *2 minutes and 51 seconds*. The longest packet dump runtime recorded was *8 minutes and 20 seconds* for throughput of 100 packets/s and the lowest was *13 seconds* for throughput of 4000 packets/s. The average packet dump runtime in this test was approximately *2 minutes and 13 seconds*. The longest delay we recorded was *1 minute and 3 seconds* for throughput of 100 packets/s and the shortest was 14 seconds for throughput of 500 packets/s. The average delay in this test was approximately *38 seconds*.

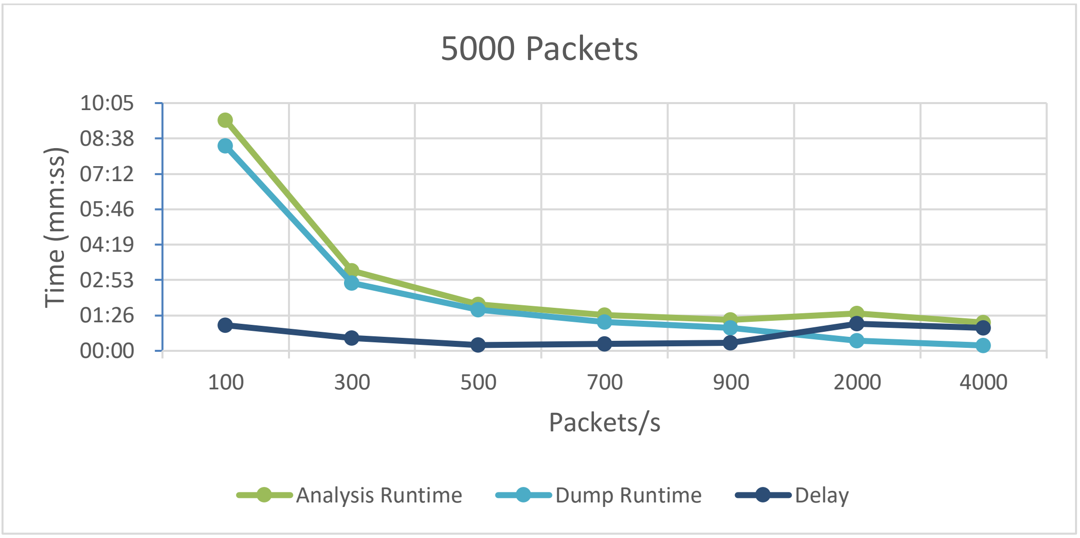


Figure 27 - 5000 Packets runtime data

Table 12 shows the remaining data collected for other variables. All the packets captured by Snort during the execution of this test were analyzed, meaning no packets were skipped or dropped during that time. Of all the packets captured, a percentage between 75% and 98% corresponded to packets whose content was in violation of the user-defined rules, while the remaining percentage corresponded to miscellaneous traffic capture by Snort.

Table 12 - 5000 Packets variable data

Packets/s Throughput	Packets Captured	Packets Analyzed	Packets Dropped	Packets Blacklisted
100	65882	100%	0%	75,89%
300	53650	100%	0%	97,17%
500	51304	100%	0%	97,43%
700	73174	100%	0%	97,69%
900	50766	100%	0%	98,48%
2000	56908	100%	0%	87,61%
4000	195924	100%	0%	97,36%

Figure 28 shows the data collected from the tests performed for 10000 packets. The longest Snort analysis runtime recorded in this test was *19 minutes and 3 seconds* for throughput of 100 packets per second and the shortest was *2 minutes and 50 seconds* for throughput of 7000 packets per second. The average Snort analysis runtime in this test was approximately *5 minutes and 53 seconds*. The longest packet dump runtime recorded was *16 minutes and 40 seconds* for throughput of 100 packets/s and shortest was *10 seconds* for throughput of 9000 packets/s. The average packet dump runtime in this test was approximately *3 minutes and 17 seconds*. The longest delay we recorded was *2 minutes and 23 seconds* for throughput of 100 packets/s and the shortest was *1 minute and 57 seconds* for throughput of 500 packets/s. The average delay in this test was approximately *2 minutes and 36 seconds*.

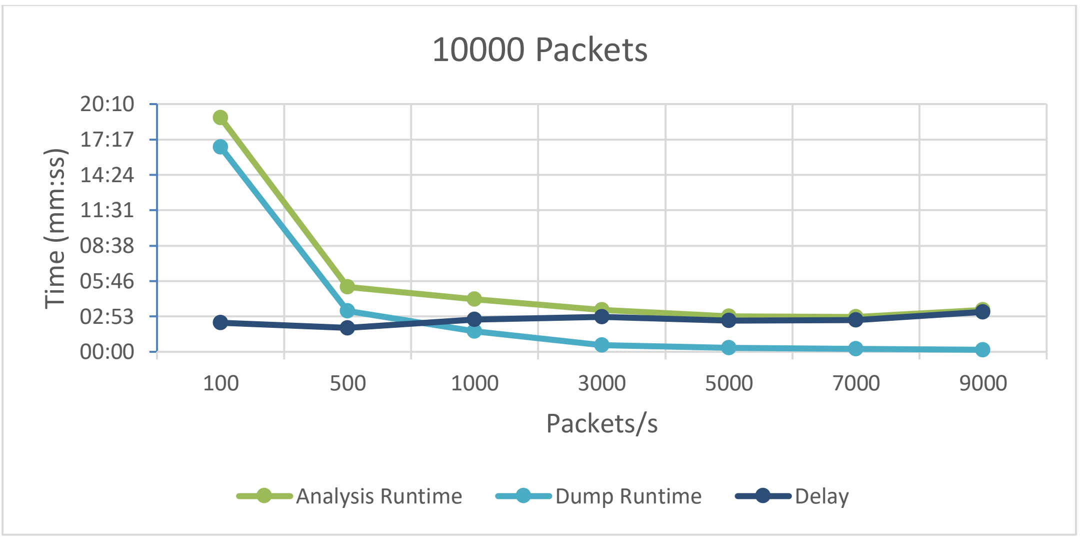


Figure 28 - 10000 packets runtime data

Table 13 shows the remaining data collected for other variables. All the packets captured by Snort during the execution of this test were analyzed, meaning no packets were skipped or dropped during that time. Of all the packets captured, a percentage between 40% and 97% corresponded to packets whose content was in violation of the user-defined rules, while the remaining percentage corresponded to miscellaneous traffic capture by Snort.

Table 13 - 10000 packets variable data

Packets/s Throughput	**Packets Captured**	**Packets Analyzed**	**Packets Dropped**	**Packets Blacklisted**
100	247000	100%	0%	40,44%
500	121334	100%	0%	82,40%
1000	106842	100%	0%	93,39%
3000	102444	100%	0%	97,19%
5000	101960	100%	0%	97,72%
7000	102255	100%	0%	97,33%
9000	104410	100%	0%	95,75%

4.3.4. Rules Scenario – Discussion of Results

In the results displayed in subsection *4.3.3. Rules Scenario – Results Presentation*, we were able to prove that the system implemented was able to detect and drop the packets accordingly, without ever showing service breaks during the analysis for high packet loads, maintaining a cadence of approximately *30 seconds* per packet without missing or skipping packets.

According to the results shown in Table 14, the analysis took the longest in the 5000 and 10000 packets per second tests when the rate of packet throughput was the lowest at 100 packets per second. With increased throughput, we saw that time decrease progressively; as we came closer to reaching rates of 1000, 5000 and 10000 packets per second, the difference between the different rates had shorten, approximately, *2 seconds* for 1000 packets test, *10 seconds* for 5000 and 40 seconds for 10000 packets test. This was caused by automatic increase of speed by Snort to adjust itself to handle the extra load without causing major delays in the analysis.

With that time decrease, the results displayed a tendency to become more consistent, which meant that the system started to stabilize at around *500 packets/s* in the 1000-packet test, *700 packets/s* in the 5000-packet test and *1000 packets/s* in the 10000 packets test. This gives a packet analysis average after consistency of approximately *5 seconds* in the 1000-packet test, *15 seconds* in the 5000-packet test and *30 seconds* in the 10000-packet test.

Table 14 - Rule Test Scenario Results Summary

	1000 Packets/s	**5000 Packets/s**	**10000 Packets/s**
Analysis Runtime High/Low	2m7s / 14s	9m23s / 1m09s	19m03s / 2m50s
Dump Runtime High/Low	1m40s / 10s	8m20s / 13s	16m40s / 10s
Delay Runtime High/Low	32s / 4s	1m06s / 14s	3m15s / 1m57s
Average Analysis Runtime	14s	2m51s	5m53s
Average Dump Runtime	31s	2m13s	3m17s
Average Delay Runtime	12s	38s	2m36s

Chapter 5 - Conclusions and Future Work

In this chapter we present the conclusions of our work and some directions for future research work.

5.1. Main Conclusions

In this thesis, we designed, developed and tested a system capable of, on the one hand, bringing transparency to IoT systems, where manufacturers declare what their devices will do (communicate) and these declarations are verifiable and, on the other, giving IoT users control over their data privacy and security, by controlling the communication of their networked IoT devices. We believe that this contributes to establishing trust between device manufacturers and their users/consumers. With the system referred to, we were able to answer the main research question posed in section *1.2. Research Question* of *Chapter 1 – Introduction*, regarding the design and implementation of a mechanism to provide the user/consumer with control of the communication of networked IoT devices.

This mechanism is composed of three main components: rule definition, incident detection and action according to incident. The rule definition phase consists of defining the intents and rules the system will comply with when filtering traffic to see if violations of those rules are happening. The detection phase consists of analyzing the contents of the packets of the network traffic that is being mirrored to Snort by checking for content that is specified in the translated Snort rules. The action phase consists of the system's reaction when the content and destination of a packet comply with its rules, or, if an intent is being violated, an alarm is triggered, according to whether the violation relates to content or to destination address. If it is a rule that is being violated, the system will drop that packet, preventing it from reaching its destination. These three main components represent the result of the research goals defined in section *1.3. Research Goals* of *Chapter 1 – Introduction.*

With this mechanism, the user/consumer knows whether or not their devices are compliant with the communication intents defined by the manufacturer, because the system will show them alerts if such violation occurs, as demonstrated in the test results in subsection *4.2.2. Test Scenario I - Results Presentation* for the section *4.2.1. Description of Test Scenario I – Violations of the manifest of intents* test scenario. With their defined traffic control rules, the user now possesses control over communication intents by monitoring exactly the amount and type of information the device is sending back to the manufacturer or some other source and by blocking information that might compromise their privacy, thus ensuring that their IoT device is secure, as demonstrated in the test results of section *4.2.5. Test Scenario II – Results Presentation* for the section *4.2.4. Description of Test Scenario II - Violations of user-defined rules*.

5.2. Future Work

As a focus of future work, we propose the integration of a component like HTTPS analyzers for understanding HTTPS content within the proposed architecture and HTTPS traffic generation, because the traffic of IoT devices is mostly encrypted.

Secondly, we also propose that when a new device is registered on the network for the first time it should confront the user with the manufacturer's intents, in the same way as older versions of Android phones used to ask the user to give the application the permissions it required to work on the smartphone that was installed on. This feature would act as a moment of validation for the environment's security policy.

Lastly, we propose implementation of a business intelligence system to gather traffic captured by Snort IPS from IoT devices to analyze infringements of intents specified by manufacturers. This Business intelligence system would compile the gather traffic information into graphics to display it on a dashboard for the IoT device owner to visualize for analytic proposes.

Bibliography

Amar, Y., Haddadi, H., Mortier, R., Brown, A., Colley, J., & Crabtree, A. (2018). *An Analysis of Home IoT Network Traffic and Behaviour*. Retrieved from Cornell University website: https://arxiv.org/abs/1803.05368v1

Bhosale, D. A., & Mane, V. M. (2016). Comparative study and analysis of network intrusion detection tools. *Proceedings of the 2015 International Conference on Applied and Theoretical Computing and Communication Technology, ICATccT 2015*, 312–315. https://doi.org/10.1109/ICATCCT.2015.7456901

Blum, S. (2019, February 21). Google Calls Hidden Microphone in Its Nest Home Security Devices an "Error." Retrieved October 14, 2020, from https://www.popularmechanics.com/technology/security/a26448907/google-nest-hidden-microphone/

Calado, J. P. da C. (2018). *Open source {IDS}/{IPS} in a production environment: comparing, assessing and implementing* (Master's thesis, Science Falculty of University of Lisbon). Retrieved from https://repositorio.ul.pt/handle/10451/35418

Chakraborty, S. (2017, October 7). When smart gadgets spy on you: Your home life is less private than you think. Retrieved January 12, 2021, from https://economictimes.indiatimes.com/tech/internet/when-smart-gadgets-spy-on-you-your-home-life-is-less-private-than-you-think/articleshow/60984623.cms?from=mdr

Coble, S. (2020, January 5). Xiaomi Security Camera Shows User Wrong Video Feed. Retrieved October 14, 2020, from https://www.infosecurity-magazine.com/news/xiaomi-camera-shows-wrong-video/

Combs, G., Ramirez, G., Harris, G., & Sharpe, R. (2016). *Wireshark - Network Protocol Analyzer*. Retrieved from https://www.wireshark.org/

El-Maghraby, R. T., Elazim, N. M. A., & Bahaa-Eldin, A. M. (2018). A survey on deep packet inspection. *Proceedings of ICCES 2017 12th International Conference on Computer Engineering and Systems, 2018-January,* 188–197. https://doi.org/10.1109/ICCES.2017.8275301

Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011). Android permissions demystified. *Proceedings of the ACM Conference on Computer and Communications Security*, 627–636. https://doi.org/10.1145/2046707.2046779

Google. (2021, August 13). Permissions on Android. Retrieved September 1, 2021, from https://developer.android.com/guide/topics/permissions/overview

Hafeez, S. (2016). Deep Packet Inspection using Snort (Master's thesis, The Islamia University of Bahawalpur). Retrieved from https://dspace.library.uvic.ca/bitstream/handle/1828/7545/Hafeez_Saad_MEng_2016.pdf?sequence=4&isAllowed=y

Iannella, R. (2004). The Open Digital Rights Language: XML for Digital Rights Management. *Information Security Technical Report, 9*(3), 47–55. https://doi.org/10.1016/S1363-4127(04)00031-7

IPSOS Institute. (2019). *CIGI-IPSOS GLOBAL SURVEY - INTERNET SECURITY & TRUST 2019 PART I & II: INTERNET SECURITY, ONLINE PRIVACY & TRUST*. Retrieved from Centre for International Governance Innovation website: https://www.cigionline.org/cigi-ipsos-global-survey-internet-security-and-trust/

Jha, A. K., Lee, S., & Lee, W. J. (2015). Modeling and Test Case Generation of Inter-component Communication in Android. *Proceedings - 2nd ACM International Conference on Mobile Software Engineering and Systems, MOBILESoft 2015*, 113–116. https://doi.org/10.1109/MobileSoft.2015.24

Julien, V., Jonkman, M., & Metcalf, W. (2007). *Suricata*. Retrieved from https://suricata.io/

Kebede, M. G., Sileno, G., & Van Engers, T. (2021). A Critical Reflection on ODRL. In V. Rodríguez-Doncel, M. Palmirani, P. Araszkiewicz Michałand Casanovas, U. Pagallo, & G. Sartor (Eds.), *AI Approaches to the Complexity of Legal Systems XI-XII* (pp. 48–61). Cham: Springer International Publishing.

Lakshminarayana, D. H., Philips, J., & Tabrizi, N. (2019). A survey of intrusion detection techniques. *Proceedings - 18th IEEE International Conference on Machine Learning and Applications, ICMLA 2019*, 1122–1129. https://doi.org/10.1109/ICMLA.2019.00187

Lee, D. (2018, May 24). Amazon Alexa heard and sent private chat - BBC News. Retrieved October 14, 2020, from https://www.bbc.com/news/technology-44248122

Lee, S. (2020). *Open5GS Open source project of 5GC and EPC*. Retrieved from https://open5gs.org/open5gs/docs/guide/01-quickstart/

LF Projects LLC. (2021). *Magma An Open-Source Platform For Building Carrier-Grade Networks*. Retrieved from https://www.magmacore.org/

Li, X. Y., Brouard, K., & Cai, Y. (2012). Dynamic policy and charging control framework. *Bell Labs Technical Journal*, *17*(1), 105–123. https://doi.org/10.1002/BLTJ.21526

Mazhar, M. H., & Shafiq, Z. (2020). Characterizing smart home IoT traffic in the wild. *Proceedings - 5th ACM/IEEE Conference on Internet of Things Design and Implementation, IoTDI 2020*, 203–215. https://doi.org/10.1109/IoTDI49375.2020.00027

McCauley, D., & Lara, V. (2018). *What the Internet of Things means for consumer privacy*. Retrieved from The Economist Intelligence Unit Limited website: https://eiuperspectives.economist.com/technology-innovation/what-internet-things-means-consumer-privacy-0/white-paper/what-internet-things-means-consumer-privacy

Morgan, J. (2021, July 30). Difference Between Explicit and Implicit. Difference Between Similar Terms and Objects. Retrieved November 30, 2021, from http://www.differencebetween.net/language/difference-between-explicit-and-implicit/

Naik, N., Jenkins, P., Cooke, R., Gillett, J., & Jin, Y. (2020). Evaluating Automatically Generated YARA Rules and Enhancing Their Effectiveness. *2020 IEEE Symposium Series on Computational Intelligence (SSCI),* 1146–1153. https://doi.org/10.1109/SSCI47803.2020.9308179

P, S. (2010). *Ostinato Traffic Generator for Network Engineers*. Retrieved from https://ostinato.org/

Peffers, K., Tuunanen, T., Gengler, C. E., Rossi, M., Hui, W., Virtanen, V., & Bragge, J. (2007). The design science research process: A model for producing and presenting information systems research. *ArXiv*. Retrieved from https://www.researchgate.net/publication/228650671_The_design_science_research_process_A_model_for_producing_and_presenting_information_systems_research

Pelteret, M., & Ophoff, J. (2016). A review of information privacy and its importance to consumers and organizations. *Informing Science, 19*(1), 277–301. https://doi.org/10.28945/3573

Quintero, B., Martínez, E., Álvarez, V. M., Hiramoto, K., Canto, J., Bermúdez, A., & Infantes, J. A. (2004). *VirusTotal*. Retrieved from https://www.virustotal.com/gui/home/upload

Roesch, M. (2016). *Snort - Network Intrusion Detection & Prevention System*. Retrieved from https://www.snort.org/

Rommer, S., Hedman, P., Olsson, M., Frid, L., Sultana, S., & Mulligan, C. (2020). 5G Core Network - Powering Digitalization. In T. Pitts & I. C. Silva (Eds.), *Academic Press*. https://doi.org/10.1515/9783110724509-008

Sears, A. (2019, September 22). “Felt so violated:” Milwaukee couple warns hackers are outsmarting smart homes. Retrieved October 14, 2020, from https://www.fox6now.com/news/felt-so-violated-milwaukee-couple-warns-hackers-are-outsmarting-smart-homes

Shah, S. A. R., & Issac, B. (2018). Performance comparison of intrusion detection systems and application of machine learning to Snort system. *Future Generation Computer Systems*, *80*, 157–170. https://doi.org/10.1016/J.FUTURE.2017.10.016

Sivaraman, V., Gharakheili, H. H., Vishwanath, A., Boreli, R., & Mehani, O. (2015). Network-level security and privacy control for smart-home IoT devices. *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2015*, 163–167. https://doi.org/10.1109/WiMOB.2015.7347956

Statt, N. (2019, April 10). Amazon's Alexa isn't just AI — thousands of humans are listening. Retrieved October 14, 2020, from https://www.theverge.com/2019/4/10/18305378/amazon-alexa-ai-voice-assistant-annotation-listen-private-recordings

Thorun, C., Vetter, M., Reisch, L., & Zimmer, K. A. (2017). *Indicators of consumer protection and empowerment in the digital world Results and recommendations of a feasibility study*. Retrieved from Institute for Consumer Policy website: https://www.bmjv.de/G20/DE/ConsumerSummit/_documents/Downloads/Studie.pdf?__blob=publicationFile&v=1

Appendices

Appendix A – PCRF and PCEF Solutions Comparison Table

	Open5gs	Magma
Hardware Requirements		
OS:	Debian/Ubuntu 18.04 or later	Linux
	CentOS	
	Fedora	
	MacOSX (Apple Silicon)	MacOS
	MacOSX (Intel)	
	FreeBSD	
RAM:	4GB min 8GB max	4GB min 8GB max
CPU:	Dual Core min quad-core max	Dual Core min quad-core max
Disk Requirements:	20 GB min 40 GB max	20 GB min 40 GB max
Additional Software Requirements		
Database:	MongoDB	MariaDB (Storage Users and Organizations)/Postgres (Model Storage)
Web UI:	NodeJs, React	Golang
Rules Creation:	Graphics User Interface	Graphics User Interface + Rest API
Policy Rule Configuration		
	In Open5Gs, the policy rules are configured individually for each device using the Web UI or using a provided python library interface which writes directly to the rules database.	In Magma, unlike the other two solutions, the policy rules are configured in a dedicated policy table where one or more IMSI of devices are associated with the policy rule and the policy rule must be associated with a network. Policies can be configured for different types of application (Facebook, WhatsApp, Ticktock, etc.) for different service types (audio, video or chat).
	The policy rules in Open5gs are defined by: • QCI (QoS Class Identifier).	The policy rules in Magma are defined by: • Policy ID.

• ARP Priority Level (1-15). • MBR Downlink/Uplink. • GBR Downlink/Uplink;	• Flows (Action, Direction, Protocol, Source/Destination IP). • Priority. • Number of Subscribers. • Monitoring Key (hex). • Rating (Service Charging). • Tracking Type;
Packet Inspection	
Open5gs does not possess built-in capabilities to detect protocols or destination address; for that we must turn to external components like Flow-Description AVP of type IPFilterRule to filter packets by: • Direction (In or Out). • Source and Destination IP Address. • Protocol. • Source and Destination Port;	In Magma, if DPI (Deep Packet Inspection) is enabled on the access gateway, protocols can be detected and filtered. Regarding content type, we can filter by application and content type together with destination address when we create the policies in the NMS.
Alarms	
Open5Gs does not possess the ability to configure alarms.	In Magma, we can configure alarms for debugging purposes to detect potential issues on the network, ordered by severity ('Critical', 'Major', 'Minor' and 'Misc'). It also possesses an alarm receiver that allows pushing of notifications of alarms in real time. Most of these alerts come loaded with the application, but the user can define custom alerts.
Metrics	
Open5gs does possess metrics capability.	Magma possesses the ability to generate metrics to provide a great deal of visibility into gateways, base stations, subscribers, reliability, throughput, etc. These metrics are stored for 30 days.
Data Dumping	
Open5GS does not possess built-in data dumping.	Magma possesses the ability to export logs for analysis of the services it provides.

Appendix B – ODRL Code Examples

```
"@type": "agreement",
"permission":
    "assigner": "OrganizationX", "assignee": "OrganizationY",
    "action": "transfer", "target": "datasetA"
```

Figure 29 - Delegation as transfer (Kebede et al., 2021)

```
"@type": "agreement",
"permission":
    "assigner": "OrganizationX", "assignee": "OrganizationY",
    "action": "grantUse", "target": "datasetA",
    "duty": [{ "action": "nextPolicy", "target": "ex:newPolicy" }]

"@type": "set",
"uid": "ex:newPolicy",
"permission": [{ "action": "read", "target": "datasetA" }]
```

Figure 30 - Delegation as s granting conditional usage (Kebede et al., 2021)

```
"@type": "agreement",
"permission":
    "assigner": "OrganizationX", "assignee": "OrganizationY",
    "action": "grantUse", "target": "datasetA",
    "duty": [{ "action": "nextPolicy", "target": "ex:newPolicy" }]

"@type": "set",
"uid": "ex:newPolicy",
"permission": [{ "action": "read", "target": "datasetA" }]

    "duty": [{ "action": "nextPolicy", "target": "ex:newPolicy" }]

"@type": "set",
"uid": "ex:newPolicy",
"permission": [{ "action": "read", "target": "datasetA" }]
```

Figure 31 - Delegation as nested granting of conditional usage (Kebede et al., 2021)

Appendix C – Intents & User Rule Management REST API Documentation

The Intent & User Rule Management REST API Documentation presents the entry points used, the information those entry points receive as inputs and the outputs they produce. The *getAllIntents* method returns *200 HTTP successful code* if there is a list of intents (array) stored in the database, *404 HTTP error code* if no intents exist in the database and *500 HTTP internal error code* if an unexpected server error occurs. The *createIntent* method receives an intent (json body) as an input and returns *200 HTTP successful code* with the UUID ID of that intent stored in the database, *400 HTTP error code* if the intent is missing or possesses incorrect parameters and *500 HTTP internal error code* if an unexpected server error occurs. The *getIntentByID* method receives the ID of the intent to be returned and returns *200 HTTP successful code* if the intent has been found, *400 HTTP error code* if the intent UUID does not correspond to a version 4 UUID, *404 HTTP error code* if no intent has been found and *500 HTTP internal error code* if an unexpected server error occurs. The *updateIntentByID* receives the ID of the intent to be updated and the intent itself (json) has an input and returns *200 HTTP successful code* if the intent has successfully updated, *400 HTTP error code* if the intent is not specified correctly, *404 HTTP error code* if the intent ID has not found and *500 HTTP internal error* code if an unexpected server error has occurred. The *deleteIntentByID* receives as an intent UUID ID as an input and returns *200 HTTP successful code* if the intent was deleted successfully, *400 HTTP error code* if the UUID entered is not a valid version 4 UUID, *404 HTTP error code* if the intent does not exist and *500 HTTP internal error code* if an unexpected server error occurs. The user-defined rules methods receive and return the same as an intent, but for a user-defined rule instead.

```
###Intents###
/intent:
  get:
    tags:
      - intent-controller
    summary: Get all intents
    operationId: getAllIntents
    responses:
      200:
        description: Get OK - Successfully got all intents.
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Intent'
        404:
          description: Get Error - No intents exist.
        500:
          description: Get - Server unexpected error.
    post:
      tags:
        - intent-controller
      summary: Create intent
      operationId: createIntent
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Intent'
      responses:
        200:
          description: Post OK - Created a new intent successfully.
          content:
            application/json:
              schema:
                type: object
                properties:
                  intentId:
                    type: string
                    format: uuid
                    example: e4768e24-80b6-11eb-9439-0242ac130002
        400:
          description: Post Error - The intent is missing or may
have incorrect parameters.
        500:
          description: Post Error - Server unexpected error.
  /intent/{intentId}:
    get:
      tags:
        - intent-controller
      summary: Get a intent by intent ID
      operationId: getIntentById
      parameters:
        - name: intentId
          in: path
          required: true
          schema:
            type: string
            format: uuid
      responses:
        200:
          description: Get OK - Successful pull of intent info.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Intent'
        400:
          description: Get Error - The UUID enter is not a version 4
UUID.
        404:
          description: Get Error - Intent not found.
        500:
          description: Get - Server unexpected error.
    put:
      tags:
        - intent-controller
      summary: Update a intent by ID
      operationId: updateIntentById
      parameters:
        - name: intentId
          in: path
          required: true
          schema:
            type: string
            format: uuid
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Intent'
      responses:
        200:
          description: Put OK - Successfully updated.
        400:
          description: Put Error - Invalid intent arguments.
        404:
          description: Put Error - Intent not found.
        500:
          description: Put - Server unexpected Error.
    delete:
      tags:
        - intent-controller
      summary: Delete a intent by ID
      operationId: deleteIntentById
      parameters:
        - in: path
          name: intentId
          required: true
          schema:
            type: string
            format: uuid
      responses:
        200:
          description: Delete OK - Successfully deleted.
        400:
          description: Delete Error - The UUID enter is not a
version 4 UUID.
        404:
          description: Delete Error - Intent not found.
        500:
          description: Delete - Server unexpected error.

  ###Rules###
  /rule:
    get:
      tags:
        - rule-controller
      summary: Get all user rules
      operationId: GetAllRules
      responses:
        200:
          description: Get OK - Successfully got all user rules.
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Rule'
        404:
          description: Get Error - No rules exist.
        500:
          description: Get - Server unexpected error.
    post:
      tags:
        - rule-controller
      summary: Create user rule
      operationId: createRule
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Rule'
      responses:
        200:
          description: Post OK - Rule created successfully.
          content:
            application/json:
              schema:
                type: object
                properties:
                  ruleId:
                    type: string
                    format: uuid
                    example: e4768e24-80b6-11eb-9439-0242ac130002
        404:
          description: Post Error -  The user rule is missing or may
have incorrect parameters.
        500:
          description: Post Error - Server unexpected error.
  /rule/{ruleId}:
    get:
      tags:
        - rule-controller
      summary: Get a user rule by rule ID
      operationId: getRuleById
      parameters:
        - name: ruleId
          in: path
          required: true
          schema:
            type: string
            format: uuid
      responses:
        200:
          description: Get OK - Successful pull of rule info.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Rule'
        400:
          description: Get Error - The UUID enter is not a version 4
UUID.
        404:
          description: Get Error - Rule not found.
        500:
          description: Get - Server unexpected error.
    put:
      tags:
        - rule-controller
      summary: Update user rule by ID
      operationId: updateRuleById
      parameters:
        - name: ruleId
```

```
          in: path
          required: true
          schema:
            type: string
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Rule'
      responses:
        200:
          description: Put OK - Successfully updated.
        400:
          description: Put Error - Invalid rule arguments.
        404:
          description: Put Error - Rule not found.
        500:
          description: Put - Server unexpected Error.
    delete:
      tags:
        - rule-controller
      summary: Delete user rule by ID
      operationId: deleteRuleById
      parameters:
        - name: ruleId
          in: path
          required: true
          schema:
            type: string
            format: uuid
      responses:
        200:
          description: Delete OK - Successfully deleted.
        400:
          description: Delete Error - The UUID enter is not a
version 4 UUID.
        404:
          description: Delete Error - Rule not found.
        500:
          description: Delete - Server unexpected error.
components:
  schemas:
    Intent:
      description: Model containing intent info
      type: object
      required:
        - deviceType
        - intentProperties
      properties:
        deviceType:
          type: string
        intentProperties:
          type: array
          items:
            type: object
            properties:
              contentType:
                type: array
                items:
                  type: string
              communicationProtocol:
                type: array
                items:
                  type: string
              destinationAddress:
                type: array
                items:
                  type: string
    Rule:
      description: Model containing user rule info
      type: object
      required:
        - deviceId
        - ruleProperties
      properties:
        deviceId:
          type: string
        ruleProperties:
          type: array
          items:
            type: object
            properties:
              contentType:
                type: array
                items:
                  type: string
              communicationProtocol:
                type: array
                items:
                  type: string
              destinationAddress:
                type: array
                items:
                  type: string
```

Appendix D – Translator REST API Documentation

The Translator REST API Documentation presents the entry points used, the information those entry points receive as inputs and the outputs they produce. The *createSnortIntent* method receives an Intent as input to convert into a Snort rule and returns *200 HTTP successful code* if the intent has been translated successfully, *400 HTTP error code* if the intent received contains empty values or incorrect parameters, and *500 HTTP internal server error* if an unexpected server error has occurred. The *updateSnortIntent* method receives an intent to be updated as input and replaces the existing Snort rules in the device file with the updated ones. This method returns *200 HTTP successful code* if the intent has been updated successfully, *400 HTTP error code* if the updated intent received contains empty or incorrect parameters and *500 HTTP internal server error code* if an unexpected server error has occurred. The *deleteSnortIntent* method receives a filename as input to delete the file that contains the Snort rules for the device and returns *200 HTTP successful code* if the file has been deleted successfully, *404 HTTP error code* if the file or directory doesn't exist and *500 HTTP invalid server code* if the file permissions are not defined on the Linux operating system. The user-defined rules methods receive and return the same as the intent, but for a user-defined rule instead. The *downloadFile* method receives the filename as input and downloads the file selected by the user in the WEB Management Interface and returns *200 HTTP successful code* if the download has been successful or *500 HTTP internal server error code* if a server error stops the file download. The *uploadFiles* method receives one or more files selected by the user using the WEB Management Interface as inputs to be uploaded into the server and returns *200 HTTP successful code* if the file or files have been uploaded successfully or *500 HTTP internal server error code* if a server error stops the upload of the files.

```
###Intent###
  /intent:
    post:
      tags:
       - intent-translation-controller
      summary: Translate an manufacturer intent into a snort rule.
      operationId: createSnortIntent
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Intent'
      responses:
        200:
          description: Post OK - Intent translated successfully.
          content:
            application/json:
              schema:
               $ref: '#/components/schemas/SnortRule'
        400:
          description: Post Error - Snort rule must not contain empty
values.
        500:
          description: Post Error - Unexpected error.
  /intent/update:
    post:
      tags:
       - intent-translation-controller
      summary: Updates an exisiting intent snort rule by replace it with a
updated one.
      operationId: updateSnortIntent
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Intent'
      responses:
        200:
          description: Post OK - Intent updated successfully.
        400:
          description: Post Error - Snort rule must not contain empty
values.
        500:
          description: Post - Unexpected error.
  /intent/{filename}:
    delete:
      tags:
        - intent-translation-controller
      summary: Deletes the snort rules file of a device.
      operationId: deleteSnortIntent
      parameters:
        - name: filename
          in: path
          required: true
          schema:
            type: string
            example: intent_tac_09951213.rules
      responses:
        200:
          description: Delete OK - File deleted successfully.
        404:
          description: Delete Error - No such file/directory exists.
        500:
          description: Delete Error - Invalid permissions.
```

```
###Rule###
  /rule:
    post:
      tags:
       - rule-translation-controller
      summary: Translate an user defined rule into a snort rule.
      operationId: createSnortRule
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Rule'
      responses:
        200:
          description: Post OK - User Rule translated successfully.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/SnortRule'
        400:
          description: Post Error - Snort rule must not contain empty
values.
        500:
          description: Post - Unexpected error.
  /rule/update:
    post:
      tags:
       - rule-translation-controller
      summary: Updates an exisiting user snort rule by replace it with a
updated one.
      operationId: updateSnortRule
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Rule'
      responses:
        200:
          description: Post OK - Rule updated successfully.
        400:
          description: Post Error - Snort rule must not contain empty
values.
        500:
          description: Post Error - Unexpected error.
  /rule/{filename}:
    delete:
      tags:
        - rule-translation-controller
      summary: Deletes the snort rules file of a device.
      operationId: deleteSnortRule
      parameters:
        - name: filename
          in: path
          required: true
          schema:
            type: string
            example: rule_imei_099512133857038.rules
      responses:
        200:
          description: Delete OK - File deleted successfully.
        404:
          description: Delete Error - No such file/directory exists.
        500:
          description: Delete Error - Invalid permissions.

###FileData###
  /{filename}:
    get:
      tags:
        - file-data-controller
      summary: Download snort rule file from server.
      operationId: downloadFile
      parameters:
        - name: filename
          in: path
          required: true
          schema:
            type: string
            example: filename.rules
      responses:
        200:
          description: Get OK - Intent/Rule Snort downloaded successfully.
          content:
            text/plain:
              schema:
                type: string
                format: binary
                example: intent_tac_XXXXXXXX.rules
        500:
          description: Download - Unexpected error.
  /upload:
    post:
      tags:
        - file-data-controller
      summary: Upload snort rule files to server.
      operationId: uploadFiles
      parameters:
          - name: files
            in: query
            required: true
            schema:
              type: array
              items:
                type: string
                example: filename.rules
      responses:
        200:
          description: Post OK - Intent/ Rule Snort file uploaded
successfully.
          content:
            text/plain:
              schema:
                type: array
                items:
                  type: string
                  format: binary
                  example: intent_tac_XXXXXXXX.rules
        500:
          description: Upload - Unexpected error.

components:
  schemas:
    Intent:
      description: Model containing intent info.
      type: object
      required:
        - deviceType
        - intentProperties
      properties:
        deviceType:
          type: string
        intentProperties:
          type: array
          items:
            type: object
            properties:
              contentType:
                type: array
                items:
                  type: string
              communicationProtocol:
                type: array
                items:
                  type: string
              destinationAddress:
                type: array
                items:
                  type: string
    Rule:
      description: Model containing user rule info.
      type: object
      required:
        - deviceId
        - ruleProperties
      properties:
        deviceId:
```

```
      type: string
    ruleProperties:
      type: array
      items:
        type: object
        properties:
          contentType:
            type: array
            items:
              type: string
          communicationProtocol:
            type: array
            items:
              type: string
          destinationAddress:
            type: array
            items:
              type: string
SnortRule:
  description: Model containing snort rule info.
  type: object
  required:
    - ruleAction
    - ruleProtocol
    - sourceIP
    - sourcePort
    - flowDirection
    - destinationIP
    - destinationPort
    - ruleOptions
  properties:
    ruleAction:
      type: string
    ruleProtocol:
      type: string
    sourceIP:
      type: string
    sourcePort:
      type: string
    flowDirection:
      type: string
    destinationIP:
      type: string
    destinationPort:
      type: string
    ruleOptions:
      type: string
```