Master thesis

A framework for the automatic security analysis of IoT device applications

Author:

Nicolas Dejon

Supervisor:
Prof. Alessio Merlo

UTC Advisor:
Prof. Dritan Nace

Co-Supervisor:
Dr. Luca Verderame

UniGe Advisor:
Prof. Roberto Sacile

A thesis submitted for the degrees of

Master of Science in Complex Systems Engineering
Laurea Magistrale in Ingegneria Informatica

at
University of Technology of Compiègne (UTC, France)
University of Genoa (UniGe, Italy)
European Master in Engineering for Complex and Interacting Systems (EMECIS)

realised at
Computer Security Laboratory (CsecLab) of UniGe

25/02/2019 - 23/08/2019

Résumé

L'Internet des Objets, ou IdO (en anglais, *Internet of Things*, ou IoT), suscite des inquiétudes quant à la sécurité de son écosystème, notamment à cause de l'omniprésence attendue de la technologie qui amplifie le phénomène de manque de moyens pour correctement évaluer ses risques cybernétiques. Ce mémoire cherche donc à soutenir la sécurisation des écosystèmes IdO d'une manière automatique et continue.

Pour cela, il propose de mieux sécuriser le niveau applicatif d'un objet connecté par la conception d'un outil de vérification de politiques de sécurité. L'outil extrait un modèle de sécurité d'une application compilée pour l'objet connecté, puis, par l'intermédiaire d'un vérificateur de modèle, s'assure que l'application démontre certaines propriétés de sécurité prédéfinies. Cela signifie que l'outil permettra de vérifier que l'application d'un objet connecté est conforme à la politique de sécurité souhaitée.

Le travail réalisé se concentre sur les applications qui sont exécutées sous le système d'exploitation RIOT-OS [8]. Le framework angr [62] est utilisé afin d'extraire le modèle de sécurité de l'exécutable. Enfin, le vérificateur de modèle PRISM [38] vérifie la présence des propriétés de sécurité dans le modèle extrait.

Mots-clés : Internet des Objets ; IdO ; Internet of Things ; IoT ;objet connecté ; application ; modèle de sécurité ; vérificateur de politique de sécurité ; exécutable ; RIOT-OS ; framework angr ; PRISM ; évaluation automatique et continue des risques de sécurité.

Abstract

The Internet of Things (IoT) raises security concerns because of the expected pervasiveness of the technology and the lack of means to properly assess the cyber risks. Hence, this thesis supports the automated and continuous securing process of IoT ecosystems.

It tackles the problem of securing the IoT device application level by providing a security policy-checker tool. This tool computes the security model of a compiled IoT device application and, by using a model checker, checks the compliance with predefined security properties. This means the tool makes it possible to verify that an IoT device application is compliant with a desired security policy.

The work in this thesis is focused on IoT device applications running upon the RIOT Operating System (RIOT-OS) [8]. The angr framework [62] is used to extract the security model from the executable of the application. Finally, the model checker PRISM [38] verifies the presence of security properties in the extracted model.

Keywords: Internet of Things ; IoT ; security model ; policy checker ; executable ; IoT device application ; RIOT-OS application ; angr framework ; PRISM ; automated and continuous securing process.

Acknowledgements

This master thesis dissertation is the final cover of a six-month internship, kicking off at the same time the final run for a Master degree, and which material has been inspired and supported by a number of actors that I would like to thank here.

Thank you to the **UTC** and the **UniGe** for giving me the opportunity to develop my knowledge and wisdom in two high quality educational environments. This European Master programme between France and Italy showed the strength and the usefulness of the bridge between two neighbouring countries, and that European collaborations work better without any border.

Thank you to **Roberto Sacile**, the UniGe coordinator, for closely following my steps during this year abroad, and thank you to **Dritan Nace**, the UTC coordinator, for allowing me to live this experience in very good conditions and for his immeasurable support and kindness, and thus, since the time he supervised my fourth-year internship at the UTC. A special thanks to him.

Thank you to **Alessandro Armando**, head of the Computer Security Lab (CSec Lab) at the UniGe, for letting me work in his laboratory and participate in the internal activities as any other collaborator.

Thank you to **Alessio Merlo**, my main supervisor and in charge of the Mobile and IoT Security Unit, and **Luca Verderame**, my technical supervisor, for their precious advise and the benevolence of their supervision, giving me full autonomy on my work. I discovered more deeper what it means to work in the cyber security field and the realisations achieved during this thesis wouldn't have been possible without their sharp eyesight.

Thank you to my closest collaborators – **Davide, Gabriel and Giorgia** – for their friendship and the authentic conversations we had, that helped me better understand the next challenge to face while in doubt for my future career. I would also like to thank the rest of the **CSec Lab members** for interesting and friendly exchanges.

Thank you to **all readers**, among them the French and the Italian **committee members**, who give value for this work. I hope the elements exposed in this thesis will benefit other works and repel the barrier of knowledge a bit further.

Finally, I believe the intercultural shine of this exchange also made flourish national and international networks that set the foundations of a peaceful, respectful and bright future. I would like to express my deepest gratitude to every single person I got to meet during all my previous exchanges abroad, and specifically this one, for a strong human bond, for a personal and mutual elevation, for a better world.

Action financée par l'**Idex Sorbonne Université – Labex MS2T**, **Erasmus+** et **UniGe**.
Financial support by **Idex Sorbonne Université – Labex MS2T**, **Erasmus+** and **UniGe**.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Smartwatches, smart homes, smart factories... *Smart* is the promise of a society fully supported by the technology, which is improving the quality of life of its actors, proposing new services to live and work in a better way, new business opportunities and costs reduction. In the last 20 years, and even more recently, a massive number of devices populated our environment, that became, all of a sudden, smart. We use these devices as sensors and actuators, interacting with the surrounding as never before, to monitor, track, act on and analyse it. As true as the 1980s internet revolution changed people's life radically and very fast to eventually reach today a point of no return, this technology has now expanded to include these devices, connecting people and devices together (anyone, anything), anywhere, anytime. This is nowadays called the Internet of Things (IoT), for which a more complete definition is given in section 3.1.

1.2 Motivation

However, the expected importance of the IoT and its connectivity raise concerns about the cyber risks. The IoT opens up security issues originating from each single component –like device failures, cybersecurity vulnerabilities or even human errors– as well as holistic issues originating from the merge of all these differently vulnerable environments in clusters of IoT ecosystems. This makes the IoT an extremely vulnerable ecosystem. Readings, exposed in 3.2.2.3, clearly showed a lack of simple and ready to use methodologies that accurately assess the cyber risks in IoT ecosystems, leaving business structures unable to secure their ecosystem(s). Because of the potential important impact a security issue can have (business disaster, or even life-threatening), it is highly needed to assess the risks in order to prepare countermeasures. Furthermore, as long as these risks can't be assessed properly, people may resist to adopt the IoT, which will not be able to show its full potential and accomplish all the promised benefits.

1.3 Thesis

In the IoT, the device is the point of connection between the physical and the cyber worlds. Securing the device is thus a prerequisite for a secure IoT ecosystem. Given the plethora of ways to secure a device, for example on the device itself or by a management unit, we focus our work on securing an IoT device application by verifying that it respects security properties applicable in an IoT use case.

Our thesis is the following: if the executable of an IoT application can show it holds some security properties, then it is possible to write a security policy that the application should

follow, and there exists a way to automate the process of verifying this security policy. We impose that the application can be retrieved under its executable form, symbols included.

Security risk assessment methodologies could then base their analysis on these results, and themselves, be automated, even partially.

1.4 Goal

Our goal is to build an IoT device-centric system which, given an IoT device application in an executable format and a set of security properties, evaluates if the application meets the security properties.

Our binary (executable) approach for IoT application security analysis means it could be applied to new and older deployed devices, to third-party provider applications, be completely independent from the source code and almost independent from the hardware (see limitations in 7.2). Analysing properties at the lower-level provides the advantage of directly catching the behaviour of the program, instead of the expected and desired one of the source code that could have drifted in the development phase, e.g. because of compiler optimisations.

1.5 Contributions

We extend previous research on security policy-checking mobile applications [4] by transferring the methodology to IoT applications. The main contributions of this work are as follows:

- We present a framework for the automatic security analysis of IoT device applications through their executable and without any use of the associated source code;
- We provide a security policy-checker tool, which is an example of implementation of the presented framework;
- We developed a security policy covering one of the currently most exploited vulnerabilities in IoT ecosystems;
- We developed a use case that shows this methodology is immediately applicable for a real-world scenario;
- We discuss further research areas whose relevance is stressed in our work by having a security policy-checker tool available, like the computer assisted security audit orientation or the software over-the-air update process.

Any concerns about other software units than the IoT device application or the IoT device's physical security are out of scope.

1.6 Methodology and action plan

1.6.1 Research methodology

The methodology for this thesis consists of a literature review in order to understand where the state-of-the-art lies and what is the current focus in the field of security risk assessment for IoT ecosystems. Given the lack of a universally accepted risk assessment methodology, the focus of the work turns towards computer assisted risk assessment, which is a promising area in the current research. Efforts are therefore put in securing the IoT devices, which are believed to be the weakest point of the IoT ecosystem, endangering all its interdependent components. Since the IoT market is expected to boom, driven by a plethora of various devices running at different scales of performances, the ecosystem needs to be protected given any system configuration.

Low-end IoT devices don't own the necessary resources to implement heavy security measures, however, the security mechanisms shouldn't be pruned because of this. Therefore, a security policy checker targeting low-end IoT devices is developed and presented in this work, importing former work from the mobile security field though the inspiration of the BYODroid tool [4].

1.6.2 Action plan

The workflow of this work follows the one used in BYODroid. From the program to analyse, an Interprocedural Control-Flow Graph (ICFG, explained in 3.2.3.2) is extracted, simplified and used by a policy checker tool. Considering the low-end IoT device problematic, it was adapted to fit applications running on top of the RIOT Operating System (RIOT-OS) [8], by the use of the angr framework [59] for the program analysis and the PRISM model checker [38] for the security properties checking. The developed model generator module extracts the security model of an application given its ICFG, which will then feed the policy checker armed with security policies. The security policies reflect the top security concerns for IoT devices. The adapted workflow is presented in figure 1.1.

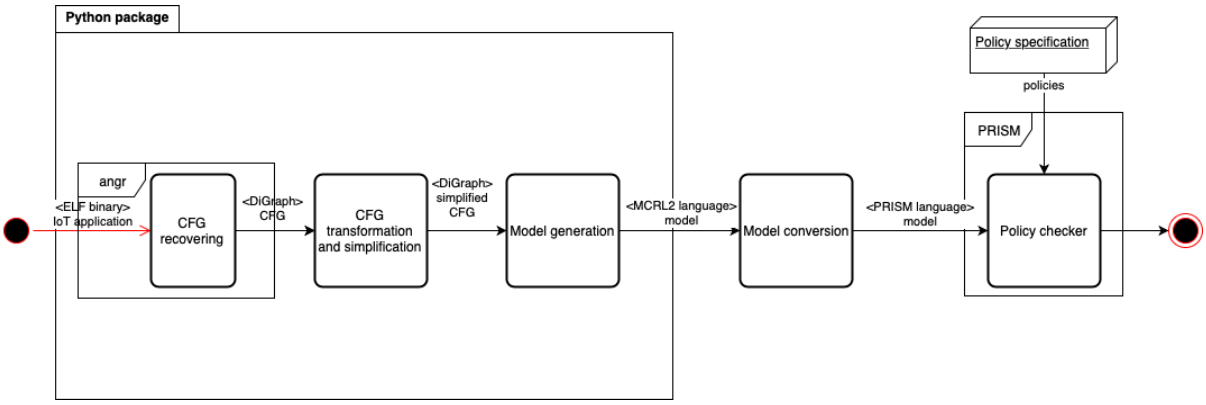


Figure 1.1 – The workflow for the low-end device IoT policy checker for RIOT applications

1.7 Overview

The rest of the thesis is structured as follows. In chapter 2 we present the working environment, settled at the computer security lab (CSecLab) of the University of Genoa (Italy). Chapter 3 introduces the research topic and presents some elements to perceive the security concerns of the IoT area, with a final focus on the security analysis of IoT device embedded applications. In the next chapter 4, the current IoT context is exposed, and two motivating examples end up in the problem formulation of this thesis. We then present, in chapter 5, the framework thought to answer the problem, which exposes the basements of the policy-checker tool explained previously, with a proposed implementation. The implemented framework is then tested against a real-world scenario in 6 by a preliminary Proof-Of-Concept study which presented positive results, therefore and thereafter tested on multiple IoT example applications for numerical evaluations. Chapter 7 discusses the achievements and the limitations of this approach. Finally, chapter 8 contains our conclusions and suggestions for further work.

Chapter 2

Presentation of the working environment – CsecLab

This master thesis has been conducted at the Università degli Studi di Genova (UniGe – University of Genoa)[63], which is directed by Paolo Comanducci. The university's origin dates back to the 13th century. Nowadays, UniGe has campuses inside the city of Genoa, but also in the nearby cities Savona, Imperia and La Spezia, and is organised into five schools, 23 departments, five libraries, 15 research centres and two centres of excellence. The University of Genoa offers bachelor's, master's and PhD courses, and first and second level university master's.

In 2014, the Computer Security Lab (CSecLab) [16], was founded and directed since then by Alessandro Armando as a university unit within the Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi (DIBRIS), the Science department of the UniGe based on the areas of Computer Science and Technology, Bioengineering, Robotics and Systems Engineering. The department itself was established in May 2012 and is currently under the direction of Enrico Puppo (Director). The mission of DIBRIS is to promote and facilitate the creation (research), transmission (education), and technology transfer of the knowledge in these areas at the national and international levels [17].

The CSecLab research interests are in the areas of Web Security, Information Hiding, Access Control and Mobile Security, recently extended to IoT Security. It also includes activities about Capture-The-Flag (CTF) challenges. The lab consists of around 15 people, including professors, PhD and master students, and external collaborators. The lab is part of the Cybersecurity Lab of the CINI (Consorzio Interuniversitario Nazionale per l'Informatica)[13], a consortium of 47 Italian public universities, which is the main point of reference for the national academic research activities in the fields of Computer Science and Information Technology in Italy. The lab is also leading the Work Task 6.1 of the Sparta project, a European Project Horizon 2020 that aims of creating a network of excellence of european cybersecurity activities, grouping 44 actors from 15 European countries.

The work has been undertaken in the CSec's Mobile and IoT Security team, under the academic supervision of Alessio Merlo and the technical supervision of Luca Verderame. The team also includes two PhD students, one external collaborator and three master students.

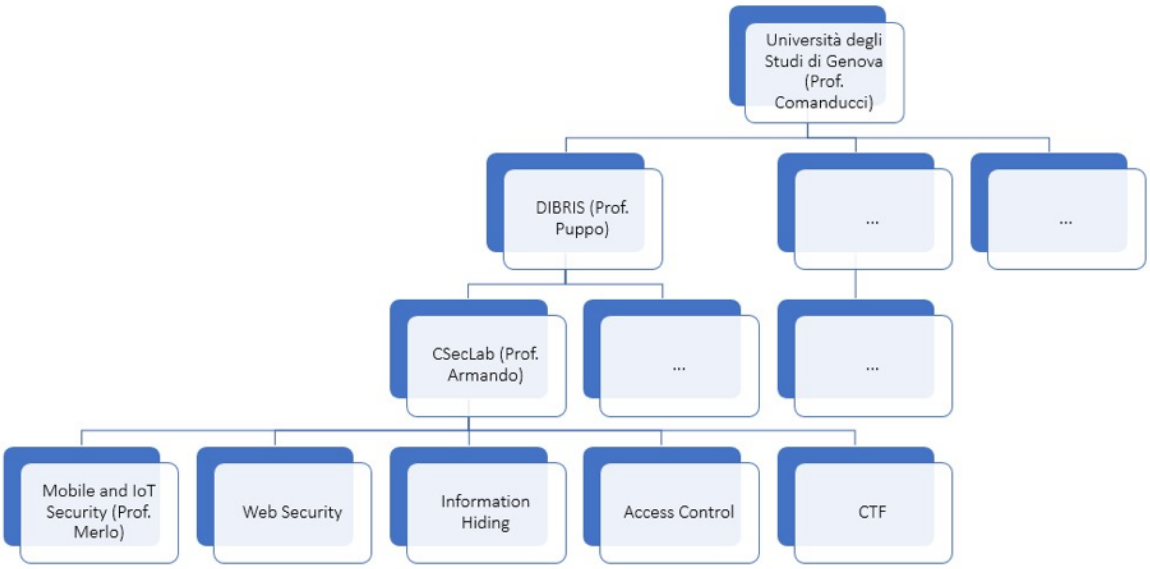


Figure 2.1 – Hierarchical view focused on the CSecLab

Chapter 3

Background

3.1 Research topic: The Internet of Things

3.1.1 What is the Internet of Things?

Before stepping into the definition of the *Internet of Things*, it is necessary to connect first to the definition of the *Internet*. The Internet is the global network of networks made up of equipment to transport information from computer systems to other computer systems, each of them connected to and reachable through the global network system. The primary goal is for humans to access documents and use applications that involve remote computer systems connected to the Internet, allowing human-human and human-machine communications.

The Internet of Things (IoT) can be seen as the expansion of the Internet to the machines, thus also supporting machine-machine communications without human intervention. These machines would be any "*thing*", including objects from our daily life (IoT was first named "Internet of Objects") : speakers, fridges, kettles, baby monitors, vehicles, pacemakers, CCTVs, smoke detectors, RFID [1] devices, factory robots... Therefore, IoT is understood as the integration of the physical world in the cyber-world, and vice-versa.

The IoT augments the "things" with intelligence, they become *smart* items. These *smart* items, interconnected with the IoT, can be remotely tracked over space and time, which basically provides an interface between virtual systems and physical systems.

Speaking in broad terms, the Internet of Things targets devices (anything) accessible by anyone, anywhere and at anytime. This supposes to have *things* connected to a backbone infrastructure in order to relay data communications using standard protocols and especially IP-based protocols. This offers a world-wide integration of the data produced by the *things*, enhancing the global knowledge accessible through the Internet.

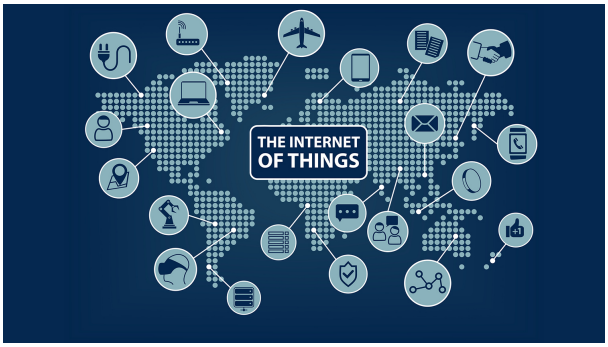


Figure 3.1 – The Internet of Things [31]

[1] Radio Frequency Identification

The IoT is a paradigm that evolved steadily since its name apparition in 1999, and according to most forecasting reports, it will become generalized in the years to come [53]. This is due to the recent popularization phase of the technology, impacting the professional and personal spheres of a growing number of users. Indeed, IoT can be used in wide range of domains including healthcare, transportation, entertainment, power grids and smart buildings...

Proofs of this popularity can be found in the creation of the Wikipedia page for *Internet of Things*, dated from the 02/07/2007 [66], with 139 648 page views as monthly average since July 2015 [65]. The French and German versions of the page already originate in October 2006, with few days between each other. The figure 3.2 shows the view of Google Trends, stating the recent popularity of the term *IoT* through the use of some keywords related to it, with a peak of keyword searches reached last year. To be recalled, popular user devices like smartphones invaded the market a bit more than ten years ago and participated to the generalisation of the technology. For example, the first Iphone by the company Apple Inc. was released on the 27th of June 2007 [3].

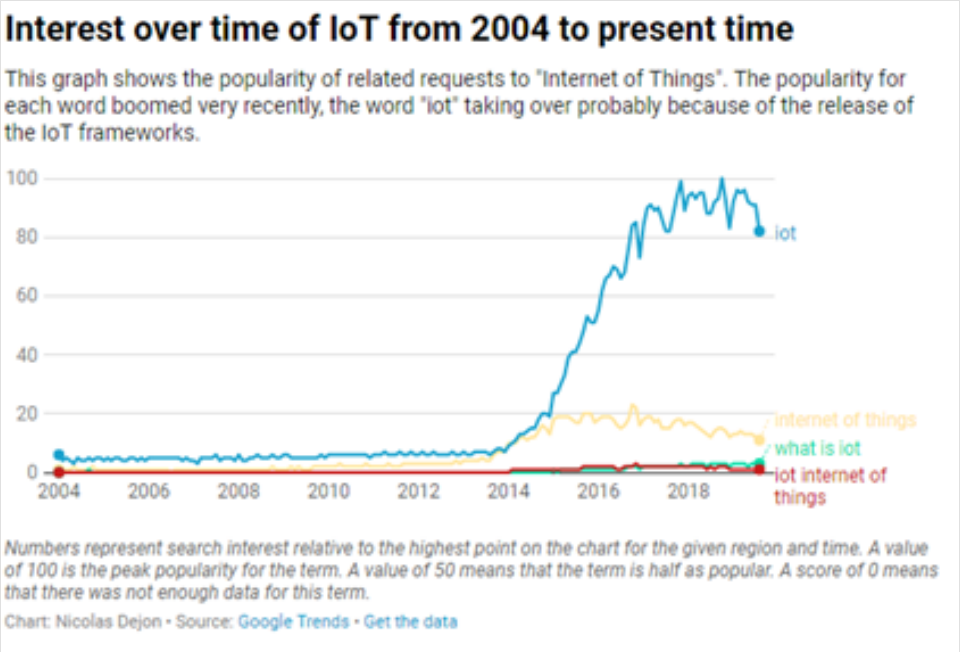


Figure 3.2 – The Internet of Things popularity [25]

The name *IoT* brings another perspective by presenting 3 visions: the *Things* oriented vision, the *Internet* oriented vision and the *Semantic* oriented vision.

The *Things* perspective emphasizes the simple, as well as complex, items that are part of the network (resource constrained devices, sensors, actuators...) that sense, live in and interact with the physical world. The European Commission is even speaking of *things* as "having identities and virtual personalities" [22]. The *Internet* oriented perspective gives the communication dimension and encourages the use of web standards as a mean for interaction, with a higher degree of automation and less human inputs. And lastly, the *Semantic* oriented perspective relates to the issues rising in the representation, the storage, the interconnection, the search and the organization of information generated by all the connected devices [6, 47].

Therefore, the IoT could be roughly defined as a world-wide network of interconnected objects, each of them uniquely addressable through standard communication protocols, that couples digital, cyber-physical and social systems (inspired by [22, 48]).

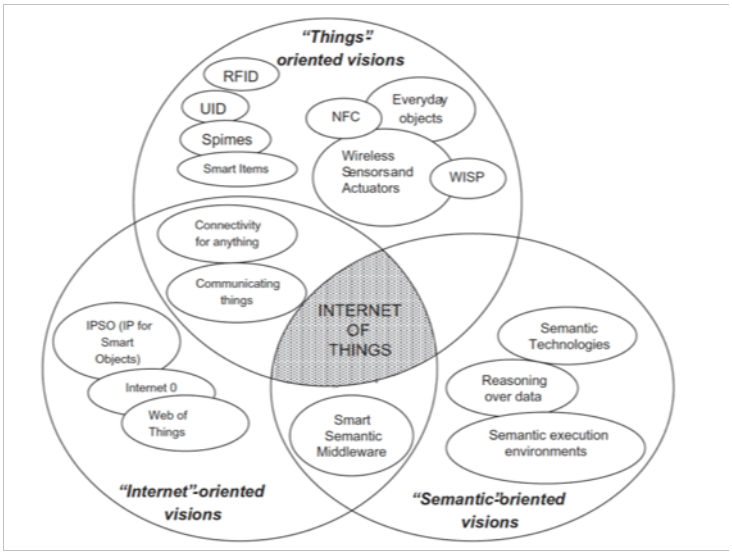


Figure 3.3 – The Internet of Things paradigm [6]

3.1.2 The IoT ecosystem

An ecosystem relates to components interrelated through their environment. The ecosystem is always dynamic and subject to disturbances. However, a durable ecosystem developed resilient and resistance properties by adapting itself to the external and internal factors, should it be dependencies, exchanges of different nature, or threats.

An IoT ecosystem is generally seen as a three-layer architecture : the Perception layer, the Network layer and the Application layer. However, no universally definition exists, and additional layers can be presented for fine-grained research [6, 56]. We use here the additional Middleware layer and expose in the following this four-layer architecture.

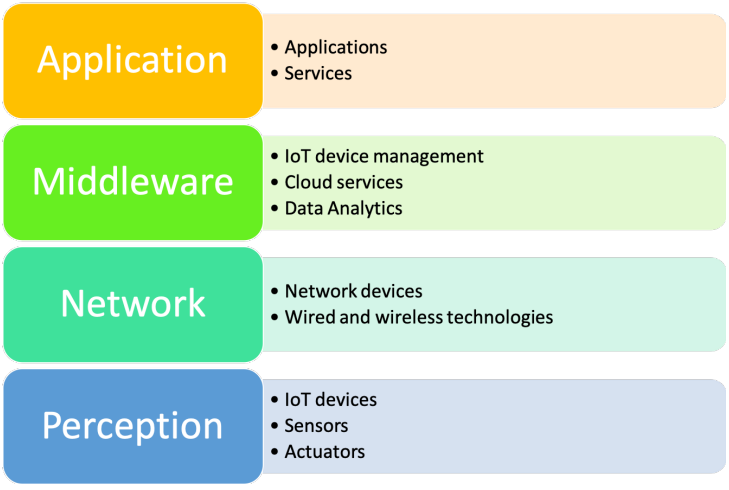


Figure 3.4 – The Internet of Things architecture

The Perception layer, also known as the *Sensors and Actuators* layer, is made of *things*. The *things* are the interface bridging the physical and the digital world by interacting with it (sensing, actuating) composed by devices, sensors and actuators. This layer collects data acquired from the physical world, processes it and uses the Network layer to transmit and receive information.

The Network layer, also called the Transport layer, is responsible of routing the data between the Perception layer and the Application layer. It is composed by smart devices, network devices (gateways, switches) and servers, using different wired and wireless technologies.

The Middleware layer provides an abstraction of the *things* and their services. This includes IoT device services and management, big data and analytics, and cloud services.

The Application layer is supported by the Middleware layer for the development of smart/intelligent applications. It exposes the system's features to the end user and gives the objective of the IoT system by providing services and applications. The rich IoT ecosystem enables the rise of new verticals, among them are smart cities, smart home, smart health, smart traffic congestion, smart water management, smart waste management, smart grid, smart parking systems and smart government. It also opens new possibilities of applications like futuristic robot taxis [6].

However, no universal definition of the system (components, architecture, model) exists for the moment and because of its heterogeneity and mix of several abstractions, it reflects the difficulty to understand the IoT concept as a whole.

3.1.3 The IoT enabling technologies

The IoT ecosystem is an agglomeration of different technologies and groups, among many others, the following enabling technologies: protocols; embedded software development; AI, machine learning and data analytics; mesh networks and peer-to-peer connectivity; cloud hosting and processing; User Interface development; database management; automation; and cybersecurity [51].

3.1.4 IoT vs M2M vs CPS

Within the landscape of IoT reside technologies like Machine-To-Machine (M2M) communications and Cyber-Physical Systems (CPS) that need to be conceptually distinguished.

The first notion, Machine-To-Machine (M2M), refers to the ability of devices to communicate with each other. It is thus possible to manage and control the devices remotely and collect machine and sensor data. However, compared to IoT, it lacks the interactions between *things* and people and is less Internet focused.

The Cyber-Physical Systems (CPS), on the other hand, is very close to the IoT since it means it has the capacity to sense and interact with the physical world. The CPS integrate devices with sensors and/or actuators. This way, the IoT brings the physical world closer to the cyber space, and *vice-versa*, with the possibility to directly interact with the environment.

IoT should be considered as an evolution or as another level of abstraction of CPS and M2M, that aggregate cyber-physical systems and systems of systems, becoming the glue between previously disconnected systems (see figure 3.5). This enables the vision of "anytime, anywhere, anymedia, anything" communications. As such, the IoT is a portmanteau word that covers any system with connected and uniquely identifiable devices, scaling up the applications offered by the Machine-to-Machine (M2M) communications and enabling the CPS to fulfill their full potential.

3.1.5 IoT vs IIoT vs MIoT

The IoT devices conquer homes and industries, differentiating themselves in the type of applications: consumer/domestic applications (wearable fitness tools, smart home thermometers, smart consumer products) which breakdowns do not immediately create emergency situations, and industrial applications (aerospace and defence, manufacturing, agriculture, health care, energy), also grouped in the so-called Industrial IoT (IIoT), which connects critical machines

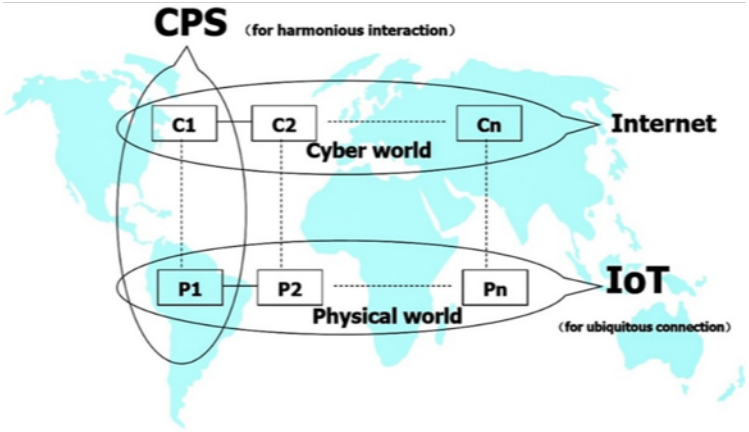


Figure 3.5 – The IoT : the glue between systems [60]

and sensors in high-stakes industries. The military applications are sometimes classified in a separate category, which is the Military IoT (MIoT) [39].

3.1.6 Security concerns in IoT

3.1.6.1 What makes IoT special?

Due to its nature and its various applications, the IoT ecosystem has specific attributes that need to be highlighted in order to understand the underlying challenges [48], also from the security point of view [57]:

Integration of the physical world As seen previously, the IoT is interacting with the real world, so cyber world and physical world are connected and can act on each other. In other words, an attack in one world could propagate to the other one, also possibly compromising legacy components that suddenly get connected and accessible from the outside;

Variability of scale in devices and systems The IoT covers a wide range of new systems and *things* of different scales, and current security solutions may have problems with scalability issues. The ecosystem could then suffer from different levels of security through the IoT vertical chain;

Dynamism of IoT IoT devices could create temporary connections fitted for a specific task or, in the contrary, persistent ones. This raises issues in device and trust management;

Variability of resource capabilities IoT devices run with limited resources implying a forced constrained regime (small memory space, low computation capability, low communication bandwidth, low power consumption), whereas cloud servers have huge memory capacity and able to respond to high computing demands. The challenge to reach the optimal configuration is always putting in competition all the components of the network;

Heterogeneity of actors devices, people, and systems are all different types of interacting actors, potentially misbehaving, also raising trust management issues. For devices and systems, this means traditional IT security measures can't be applied anymore;

Large volume of data The collection and analysis of a large volume of data coming from the *things* may expose dangerously the user's privacy. Indeed, it has been demonstrated that private information can be inferred by readings from the sensors : taking a shower, cooking, presence at home, number of people in the house...

High production demand IoT is still a new trend, that needs to be supported by a high production pace, with companies thus cutting off security expenses, and with no feedback available of the long term. Some companies may also not produce patches or updates for their devices to keep up with the production;

A new paradigm Bruce Schneier, a renowned computer security professional, claims that all the distributed nodes of the IoT ecosystem participate to sense, think and act on the physical world, which is the definition of a robot. Hence, the humans are conceptually interacting with a "world-size robot" without noticing it. This raises other questions that need contributions from other fields of study, like philosophy, to better understand the new world created and influenced by the IoT [40];

Society changes The IoT is also contributing to the changes in the professional landscape, like job creations and losses, due to the digitalization of work processes and environments [44].

3.1.6.2 Security goals

IoT systems should provide all computer security goals, which are confidentiality, integrity and availability (the CIA triad).

Confidentiality This goal ensures that private or confidential data is available only by authorized users (human or systems). This notion also covers privacy which ensures that users have control over their private information (data collection, storage, analytics).

Integrity The data crossing through the different IoT layers should flow as intended (specified and authorized changes only, no unauthorized modification or destruction of data). This notion also covers system integrity, so devices protected from manipulation operations and which behave as intended.

Availability Users of IoT applications should be able to access their data and services at all time. If not, service quality degrades the experience of the user, critical tasks could be interrupted temporary or permanently and lead to safety issues, and an attacker could take the opportunity to inject malicious code.

In addition to that, the IoT should also integrate principles as safety, resilience and reliability [42] and should propose all these solutions in a lightweight manner [68].

3.1.6.3 End-to-end security

The IoT security principles are needed to construct a system with end-to-end (E2E) security. It secures the path across all layers and components of the ecosystem and brings trust among all actors in the ecosystem. This is achieved by using proper cryptography primitives and protocols.

3.1.6.4 Some IoT attacks

From the security standpoint, the IoT ecosystem brings with it the security concerns of each of its components but also creates some specific system-level ones. This suggests that IoT increases both the attack surface and the attack capabilities of attackers.

Some important attacks at different layers can be found in the following list:

Malicious code injection attack An attacker could intentionally write a program that causes security breaches, like confidentiality damage by taking control of a device, data destruction, or illegal access to the network possibly infecting other nodes around;

(Distributed) Denial-of-service attack A denial-of-service (DOS) attack consists of making a service unavailable for the users. It could typically be because of an excessive amount of requests that can't be handled properly or an attacker that exploits a vulnerability and stops the service. The distributed DOS (DDOS) is a DOS which originates from many sources. In such case, the attack will continue as long as all sources are not blocked;

Side-channel attack The goal of this attack is to gain information by the implementation of security mechanisms, without questioning the robustness of the theoretical procedures (e.g. monitoring power consumption or memory usage, and deducing parameters like the key length or the algorithms used);

Eavesdropping attack Trying to steal information that transit on the networks is known as eavesdropping or sniffing attack, which puts in danger the users' privacy but also a possibility to conduct other types of attacks like the man-in-the-middle attack (where an intermediate compromised node is put into the ecosystem);

Physical damage By invading the physical environment, IoT devices are easily accessible and could suffer from physical attack, like the destruction of the device or sensor manipulation (e.g. placing a heating appliance next to a temperature sensor);

Sleep deprivation attack Because IoT devices run with constrained resources, they try to have aggressive power saving modes by hibernating as much as they can. This type of attack sends just enough requests to not let the device reach the hibernation phase, reducing therefore the device's lifetime, and finally makes it run out of battery and thus unresponsive.

Many other attacks can be listed in the case of IoT because of its diverse ecosystem. This will probably lead to more harmful attacks on IoT and IT systems in the future that are at the present time not possible to identify, detect and counter with current techniques and methodologies.

These attacks already revealed to be real harmful threats. For example, at the end of 2016, the Mirai Botnet, which controlled and coordinated a large number of infected devices, launched Distributed Denial of Service (DDoS) attacks which led to parts of the internet going down and affected important companies like OVH, Facebook, Twitter, Github, Netflix, CNN, Reddit [34]. Mirai is infecting IoT equipment, like security cameras and routers, with easy breakable credentials, is self-infecting, so spreading very fast on the network, and can take control of more than 50 000 devices at the same time. Since its revelation, the source code has been replicated to conduct several attacks [33] of the same nature.

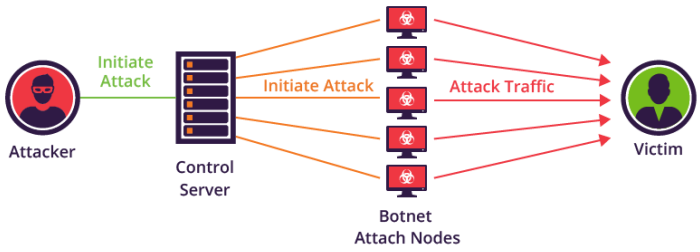


Figure 3.6 – A DDOS botnet attack explanation diagram [2]

3.1.7 Low-end IoT devices characteristics

IoT devices share the resource constraints characteristic. However, on a fine-grained analysis, they can be differentiated in high-end devices and low-end devices. The distinction is to be seen in terms of resource usage and cost.

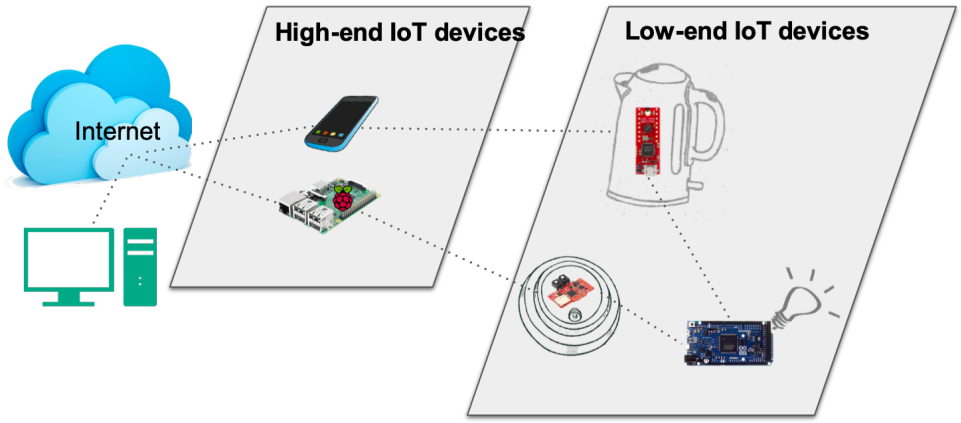


Figure 3.7 – The IoT high-end and low-end devices landscape [21]

Low-end IoT devices have minimalistic performances, between the RFID and a nano computer, and are composed by three core components:

- A microcontroller (MCU): usually single-core, few MHz clock speed, no MMU [2];
- External devices: sensors, actuators or storage, connected via UART[3], SPI[4], I2C[5]...;
- One or more network interfaces connecting the device to the Internet, usually though low-power transmission technology.

Despite low performances, low-end IoT devices need to support a broad selection of expected technologies, leading to the following OS requirements:

- Network interoperability:
  - low-power wireless technologies: IEEE 802.15.4[6], Bluetooth Low-Energy (BLE[7]) or LoRa[8];
  - expected Internet connectivity: 6LoWPAN[9], IPv6[10], UDP[11], CoAP[12];
- System interoperability: support of diverse hardware (8-bit AVR, 16-bit MSP430, 32-bit Cortex M...) easily interfaced;
- Security and Privacy: IoT will penetrate both private lives and industrial processes, they should propose adapted cryptography primitives and transparency concerning end-user data handling.

The resource limits imposed to low-end devices makes it difficult to deploy them with some widely used OSes, e.g. Arduino or Linux. Bare metal development could be suitable in some cases, yet, with the expected number of connected devices and the fast pace of innovation in the field, it would be difficult to secure an ecosystem on an IoT device case-by-case basis.

Because of these low-end devices, the remarks given earlier about the resource constraints are even more true in such case. IoT devices will thus be more or less protected, because of the constraints or just a lack of security implementation, and we will see different levels of security for different parts in a single IoT systems. The least secure device becomes the most vulnerable point-of-entry for attackers and it determines the overall level of security of the IoT systems because once a device is compromised, other devices may be exploited as well. Protecting the device is thus of higher importance.

[2] Memory Management Unit [3] Universal Asynchronous Receiver Transmitter [4] Serial Peripheral Interface [5] Inter-Integrated Circuit [6] Low rate wireless personal area network [7] Bluetooth Low Energy [8] Long Range [9] IPv6 LoW Power wireless Area Networks [10] Internet Protocol version 6 [11] User Datagram Protocol [12] Constrained Application Protocol

	Energy consumption	Memory	CPU speed	Cost
High-end IoT devices	Watt	GBytes	GHz	medium-high
Low-end IoT devices	mWatt	kBytes-MBytes	MHz	very low-low

Table 3.1 – Low-end IoT devices and high-end IoT devices comparison table

3.2 IoT device application security analysis

3.2.1 Cyber security risks of IoT device applications

3.2.1.1 Definitions of risk

As defined by the National Institute of Standards and Technology (NIST), risk is "*a measure of the extent to which an entity is threatened by a potential circumstance or event, and typically is a function of: (i) the adverse impact, or magnitude of harm, that would arise if the circumstance or event occurs; and (ii) the likelihood of occurrence*" [46]. Generally speaking, we consider an event to be of extreme high risk if there is a high impact and a high likelihood to happen, and oppositely coupled a very low risk if it has a low impact and a low likelihood. Hence, the formula: $Risk = Likelihood * Consequences$ [26].

In a blended view of relevant literature [19, 26, 46], a risk can be related to or characterized by :

- a scenario: it's origin ; a certain activity, event or incident (i.e. threat) ; a specific reason for its occurrence ; time and place of occurrence;
- the probability of the scenario to occur;
- its consequences, results, damages or impact;
- protective mechanisms and controls (together with their possible lack of effectiveness).

Information used to estimate impact and likelihood usually comes from:

- past experience or data and records (e.g. incident reporting);
- reliable practices, international standards or guidelines;
- market research and analysis;
- experiments and prototypes;
- economic, engineering or other models;
- specialist and expert advice.

3.2.1.2 Definition of cyber security risk

When speaking about cyber security, risk is about threats: the exploitation of vulnerabilities by threat actors to compromise device or data confidentiality, integrity, or availability [46].

3.2.1.3 Why are IoT device risks different from IT risks?

The importance of the role of the IoT device within the IoT ecosystem is competing with its capacities.

NIST [46] defines 3 considerations:

1. Many IoT devices interact with the physical world in ways conventional IT devices usually do not.
   (a) There are uncertainties associated to the measurements of the physical world from IoT sensor data;
   (b) The devices can affect the physical world by its ability to make changes to physical systems through actuators (endanger human safety, damage or destroy equipment and facilities, or cause major operational disruptions);
   (c) The IoT network interfaces often enable remote access to physical systems that previously could only be accessed locally.
2. Many IoT devices cannot be accessed, managed, or monitored in the same ways IT devices can.
   (a) Many IoT devices often seen as "black boxes";
   (b) Little or no access to and management of their software and configuration (no access to the internal workings, the partner company could stop distributing patches...).
3. The availability, efficiency, and effectiveness of cyber security and privacy capabilities are often different for IoT devices than conventional IT devices.
   (a) Many IoT devices don't or cannot support the range of cyber security and privacy capabilities typically built into conventional IT devices (no logs of cyber security and privacy events, or give no access to it);
   (b) If there is a centralized IoT device management, the maintenance capabilities can be very excessive;
   (c) Some post-market capabilities for conventional IT, such as network-based intrusion prevention systems, anti-malware servers, and firewalls, may not be as effective at protecting IoT devices as they are at protecting conventional IT;
   (d) IoT devices often use protocols that cyber security and privacy controls for conventional IT cannot understand and analyse.

To that, we can extend the remarks with the limited resources available on the devices, that can't afford to embed anti-virus or anti-malware detection programs.

Its specificities make it very challenging to identify and track misbehaving actors, as well as understand and estimate the propagation of risks across the ecosystem. Threats may thus originate from physical or cyber-attacks and risks can also include damage of physical assets, loss of property and even threat to human life. To prevent these risks to occur, they need to be assessed and treated.

However, national laws are struggling to follow the pace of change in the cyber space, even with noticeable efforts from the European Union and few countries in the world, meaning regulations may be years behind systems as progressive as the IoT (the minimum security control requirements are outdated).

3.2.1.4 OWASP security measures

The Open Web Application Security Project (OWASP) [49] is an organisation focused on improving the security of software. It provides free materials for manufacturers, developers and consumers to raise their awareness about security issues when developing or using some technologies. The OWASP IoT Project is a recent additional project specifically highlighting issues in IoT.

By opening the IoT focus in a separate project, the community recognises thus the specificity of the IoT security issues and addresses them by providing recommendations and an updated list of the top 10 IoT vulnerabilities (appendix B).

3.2.2 Security risk assessment for IoT device applications

3.2.2.1 Definitions of risk assessment

A risk assessment considers the probability of a threat agent exploring a vulnerability in an asset, turning a threat into an incident that represents an impact [42].

Risk assessment is generally understood as the three-step process of identifying (risk identification), estimating (risk analysis), and prioritizing risks (risk evaluation) to organizational assets and operations [20, 48].

Risk analysis techniques include [20]:

- interviews with experts in the area of interest and questionnaires;
- use of existing models and simulations.

According to NIST, risk assessment is when the organization understands the cyber security risk lying in organizational operations (including mission, functions, image, or reputation), organizational assets, and individuals.

In a less strict manner, risk assessment can also be defined as the process of investigating possible losses using a combination of known information about the situation, and judgment about the information that is not known [5].

Risk assessment lies the foundation of treating the identified risks (*risk acceptance* when the organization considers the risk at an acceptable level; *risk mitigation* to reduce the risk using security controls; risk transfer through cyber insurances; or *risk avoidance* through removing the risky asset) [48].

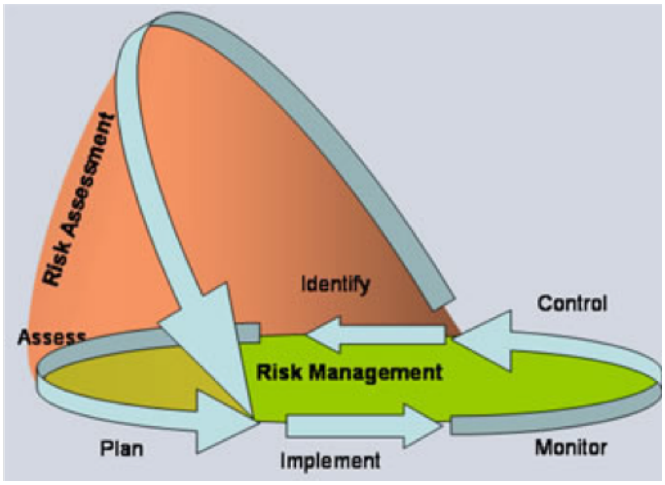


Figure 3.8 – Risk Management as a continuous process [23]

It is a part of the overall Risk Management process, which is aiming at an efficient balance between realizing opportunities for gains while minimizing vulnerabilities and losses. It is conducted in several iterations in order to fine-grain the risk analysis, as part of an overall continuous process that maintains ongoing awareness of an organization's risk environment. Management decisions are then taken based on the outcomes. However, in many organizations today, particularly Small-to-Medium-sized Enterprises, security is viewed as a single point to be reached, and not as a continuous activity. These perceptions view systems and security as static items and confirmed reports of underinvestment in cyber security, which could lead to undetected or incorrectly assessed risks, leaving the system unprotected [1].

3.2.2.2 How risk can be computed

As presented previously, risk assessment requires the calculation of probabilities and impacts.

The probability is influenced by two risk elements: threat agent motivation and vulnerabilities. If a threat agent is motivated to attack the system or if an asset is vulnerable, the probability increases. At the opposite, implemented security controls help to decrease the probability.

The impact value is directly influenced by the assets value that are items of any value, whether it is tangible like components of the system, the system itself or intangible like the organization's reputation.

3.2.2.3 Existing risk assessment methodologies and differences

Currently, numerous methodologies are available in order to perform risk analyses. In order to find the most appropriate methods for an organization, they can be classified in different areas, like the number of resources needed to conduct the analysis (people, time, equipment), the number of problems it addresses, how the analysis helps to take decisions based on the identified risks, and the metrics used (quantitative, qualitative and semi-quantitave) [9].

The consequences may be expressed in various terms of monetary, technical, operational and human impact criteria.

However, some consequences are almost impossible to quantify, like the intellectual property of digital information. Therefore, the economic value of digital assets has to reflect their economic functions before their value can be properly assigned [26].

The security risk assessment has been clearly defined in many academic papers, but there is still a reasonable degree of flexibility in their application, leading to multiple ways for conducting risk assessments though different methods, guides, and tools. They are context dependent and suitable for different types of organizations and their objectives.

As stated in [48], typically 2 approaches are developed: asset-oriented (grounded around critical assets and the harm that might occur to them, like the NIST approach) and threat-oriented (grounded around threats and how feasible they are, like OCTAVE or ENISA, that better reflect the current threat landscape). A non-exhaustive list of some of the most relevant risk assessment standards and policies can be found in [26, 58, 52, 48, 47]. Because a full review of these methodologies are unimportant for the discussion, some elements are given to the reader in the appendices A in order to appreciate the diversity and the number of a sample of all these methodologies.

Specific IoT risk assessments' methodologies Very few methods exist specifically for the IoT and focus more on automated and mathematical approaches. This is opposed to process-driven techniques like NIST and OCTAVE, which could lead to outdated assessments because of the exponential evolution of the IoT field. For example, most used methods are:

- IoTRiskAnalyzer: quantitative, framework which formally analyses IoT risks using probabilistic model checking;
- NISTIR 8228: focused at the IoT device level and looking for compliancy, collection of standards and guidelines, only scope privacy and cybersecurity risks (no safety, reliability, and resiliency orientations);
- GSMA IoT Security Assessment: framework, checklist addressing services and endpoints.

Many IoT guidance documents also exist, among them: BITAG: Broadband Internet Technical Advisory Group (BITAG), "Internet of Things (IoT) Security and Privacy Recommendations" ; CSA1: Cloud Security Alliance (CSA) Mobile Working Group, "Security Guidance for Early Adopters of the Internet of Things (IoT)"; CSA2: CSA IoT Working Group, "Identity and Access

Management for the Internet of Things"; CTIA: CTIA, "CTIA Cybersecurity Certification Test Plan for IoT Devices, Version 1.0" ; ENISA, "Baseline Security Recommendations for IoT in the context of Critical Information Infrastructures"; IIC: Industrial Internet Consortium (IIC), "Industrial Internet of Things Volume G4: Security Framework"; IoT Security Foundation (IoTSF), "IoT Security Compliance Framework, Release 1.1" ; OTA: Online Trust Alliance (OTA), "IoT Security & Privacy Trust Framework v2.5" ; UKDDCMS: United Kingdom Government Department for Digital, Culture, Media & Sport (DCMS), "Secure by Design: Improving the cyber security of consumer Internet of Things" ; OWASP: Open Web Application Security Project, "IoT Security guidance".

3.2.3 Program analysis

A program analysis offers techniques to automatically analyse the behavior or the effects of a set of statements in computer programs. This analysis can be conducted either statically (static analysis) by predicting these behaviors and effects without executing the program, or dynamically (dynamic analysis) by executing the program on a real or virtual processor.

The inputs for static analysis can be the source code, intermediate representations (IR), assembly code, or even binary code. The latter case interests us the most in this dissertation.

3.2.3.1 Kripke structure

A Kripke structure is used in model checking to represent the behaviour of a system. It is an oriented graph with nodes representing the accessible states of the system and with edges representing the transition between these states. To each state is associated a set of atomic logical propositions true in this state.

The Kripke structure can be seen as a state model of a system. The labelled nodes are the different states a system can take, and the paths in the graph are all possible executions of the system determining its behaviour. By unfolding this structure to map an execution in time, so making a state correspond to a moment in time, the Kripke structure becomes a tree. Each branch being possible execution paths (see Figure 3.9).

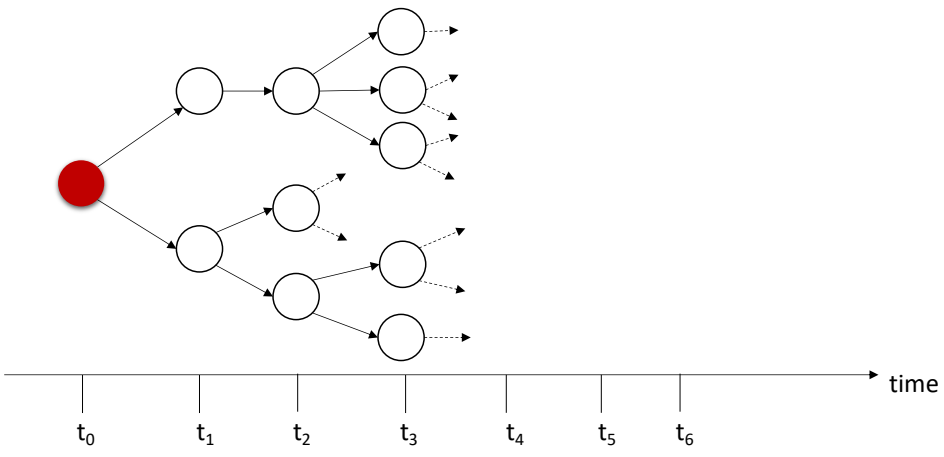


Figure 3.9 – A Kripke structure projected on a timeline

To express change in the atomic propositions over time, as would do variables during the execution of a program, one can use temporal logic and a Kripke structure as model.

3.2.3.2 Control-flow graph

One of our hypotheses is to have an IoT device application in the executable format. Even with no access to the source code, we still need to understand the program in order to extract a model out of it. The process of working backwards and understand from a system how it was originally designed is a form of reverse-engineering.

We ultimately target to retrieve the program that created the executable. One way to reverse engineer a binary software is to lift machine code with a disassembler to a higher level language, usually an Intermediate Representation (IR). From the IR, we can then deduce the structure of the program. A control-flow graph (CFG) is such a representation, which exposes all execution paths of a program procedure as explained in the following.

A CFG is a directed graph composed by nodes representing basic blocks, e.g. a piece of code, chained through edges to represent the control flow transfer.

Each basic block start is the result of a jump and another jump ends the basic block. The CFG construction algorithm thus starts at an entry point, whether it is a function or the start of the program, and every time a jump is identified, the current basic block is terminated and the algorithm continues at the jump target until the complete CFG is retrieved. The CFG specifies all possible execution paths so the basic block can be terminated by an unconditional jump, i.e. one edge, or a conditional jump, i.e. two edges (see figure 6.1).

CFG recovery is widely covered in the literature [67, 32, 35] and still challenges researchers today.

A CFG should be distinguished from a call graph. The latter represents the calling relationship between program procedures, so which function is called by or is calling other ones, but doesn't represent the control-flow.

Lastly, the CFG represents the control-flow of a single procedure. To represent the control-flow of a whole program, we use the Interprocedural CFG (ICFG). The ICFG combines all CFG from each procedure by linking the call sites of each procedure with the CFG of the called nodes. The pseudo-code of a ICFG construction algorithm is presented in Algorithm 1.

Algorithm 1 Compute the ICFG from local CFGs

```
1: procedure ICFG CONSTRUCTION
2:     app_icfg ← []
3:     start_node ← get the entry node
4:     callees_nodes ← get callees and callees of callees
5:     add local CFG of start_node to app_icfg
6:     for callee_node ∈ callees_nodes do
7:         add local CFG of callee_node to app_icfg
8:     end for
9:     return app_icfg
10: end procedure
```

Each call site node is a root node of its own local CFG and they are only referenced in the other computed local CFGs when called. As a consequence, the app_icfg contains a list of all local CFGs rooted at the $start\_node$ and each $callee\_node$, each bonded to another by reference.

Hence, by constructing the ICFG from an executable, we get the structure of the program. We can interpret it as a Kripke structure for our model checking, since it has nodes holding atomic program statements and the edges representing possible states transitions, representing all execution paths. From the retrieved CFG, it is thus possible to represent the program and extract a model of it, which will be composed of the plain control flow.

3.2.3.3 Static analysis with source code

The compiler invokes 3 parts to produce an executable:

1. Front-end: lexical analysis, parser, semantic analysis. It builds the Intermediate Representation of the program;
2. Middle-end: analyser (builds the control flow graph of each function and call graph of the program) and optimiser;
3. Back-end: CPU architecture specific optimizations and machine code generation.

The, for example, it would be possible to start from a source file and use a C compiler, e.g. clang, as a front-end to retrieve the IR level. The IR can afterwards be analysed by opt (an optimiser and analyser) which generates the CFG.

3.2.3.4 Static analysis with binary file

The binary code of a binary program can be converted to assembly instructions, then be disassembled and lifted to an IR. The IR is then analysed, and the CFG and a call graph computed.

3.2.3.5 Dynamic analysis with binary file

Dynamic analysis evaluates a program by executing it and can be used, among others, for program profiling, program debugging, and dynamic data flow analysis by using symbolic execution to capture the control flow graph, more accurately than the static analysis. However, it doesn't imply a complete coverage of the application depending on the capacity of the analyser to explore all possibilities.

3.2.3.6 Dynamic analysis with binary file and source code

If binded with the source code, one can explore the calls discovered during the dynamic analysis directly in the source files.

For example, *Valgrind* is a virtual machine that emulates processors. It groups several tools, such as *Memcheck* (use of initialized memory, reading/writing memory after it has been freed, reading/writing off the end of malloc'd blocks, memory leaks), *Cachegrind* (cache profiler) and *Callgrind* (callgraph analyser and profile the time spent in each function). The outputs of *Callgrind* can be visualized in the GUI *KCachegrind* (also used for *Cachegrind*) so to retrieve the complete callgraph of the program, from the OS to the application.

3.2.3.7 Motivations for the use of binary code static analysis

Many reasons can motivate the use of binary static analysis, as explained in [70], which has been reorganized here:

- The source code could be unavailable: this can be the case with malicious code or untrusted code, that are usually not released with their source code. It could be also because of commercial reasons such as intellectual property when a company decides not to disclose the source code. In the contrary, usually the binary code is almost always accessible, which means the analysis could be applied to a large number of programs;
- Because of the compilation phase: compilers optimisations or unintentional erroneous compiling could result in an executable which behavior differs from the one specified in the source code. The abstraction of high-level programming languages implies that the programmer's intentions may not be reflected into the running program;

- Security injections at the binary-level: voluntary security checks could be injected in the binary form to armor the final version, producing changes compared to the source code version, and thus modifying its behavior;
- Untrusted source code: even if the source code is released, it is not sure it was exactly that one which was used to create the attached binary.

However, binary code analysis is a complex operation, and it is still a major challenge for it to be accurate. This is due to the numerous and complex instructions used in modern computer architectures, but also because of the lack of abstraction representation in the binary. Indeed, it could be because a single function in the source code is split in several jumps (but represented in several basic blocks in the CFG), or that the memory space is not representing exactly the buffer abstraction, or the custom data types are drowned into plain registers and memory loosing the abstraction capability, or even because the code has been obfuscated. Also, since the code is analysed after some optmisations, whether it comes from the initial compiling phase or when lifting the binary code to the IR level, the retrieved graph could already be simplified compared to a parsed one from source, meaning that some unreachable functions could have been deleted or that the retrieved flow wouldn't represent exactly the source code.

Chapter 4

Problem definition

4.1 Context

4.1.1 The evolution of the IoT market

The pace of change and the associated consequences will be fueled by the number of connected devices expected to explose in the next few years. Indeed, the research and advisory company Gartner, which closely follows the trends in IoT, expects a high market adoption by 2023. Gartner calculated that 8.4 billion IoT *things* were in use in 2017 and forecasts 14.2 billion connected *things* in 2019 (almost the double in two years) and will eventually reach 25 billion by 2021 [24].

This substantial twist in the IT market may revolutionise the business ecosystem: costs reduction through the collected data, new business opportunities, and new services promotions. However, IoT products and services are disadvantaged compared to non-connected devices, because of the concern over cyber risks. The annual losses are evaluated from 200 billion to 1 trillion dollars worldwide [51].

4.1.2 "The 'S' in IoT stands for Security"

4.1.2.1 Why current risk assessments may fail with IoT

The research community is particularly concerned about the IoT security topic, given the number of devices to be connected and the lack of effective measures to secure the IoT ecosystem.

According to [48], four considerations make current risk assessments fail:

Shortcomings of periodic assessment While the Risk Management is a continuous process, the risk assessment phase is usually periodic, which isn't adapted to the fast evolution of an IoT in constant redefinition, with assessments outdated before they are conducted. For them to be effective, they should anticipate the possible changes occurring before the next assessment.

Changing boundaries, limited systems knowledge However, this anticipation could be almost impossible to consider because of a potentially high pace of change. Thus, the risk assessment would need to be conducted with a limited system knowledge, with the difficulty of abstraction that it may represent.

The challenge of understanding the glue Because of the structure of its ecosystem, the IoT will intensively use communications between all the components, seen as the glue. This could have undesired consequences, as effects on one part of the system could propagate and impact other parts in an undefined and unpredictable way, depending on how the data is received and processed.

Failure to consider assets as an attack platform In current risk assessment methods, the assets are to be protected and orient the risk management. However, some of these assets could be malicious and become possible attack platforms.

Because of the plethora of risk assessment methodologies, standards and guidelines needed to be combined in order to have the most accurate risk overview on a company's system, the companies could be easily lost in trying to use the most suitable one for their needs, also considering all the different manners to compute the risk. This has a direct influence on the effectiveness of the attacks' prevention and countermeasures.

Hence, at the moment of writing and under the exposed concerns, no universally adopted standard exists for specifically IoT security risk assessment. This endangers the whole ecosystem, which is at risk of failures and data manipulations that could lead to business disasters, user threats or even life threats.

4.1.2.2 Additional concerns

All the enabling technologies used in the IoT ecosystem (big data, cloud, Artificial Intelligence...) add additional difficulties to assess risks accurately since they enlarge the attack surface of the system. Because the IoT is still in its infancy stage, it is expected to witness vulnerabilities or attack methods that are undiscovered as yet. This shows the huge potential harmfulness of not being able to construct attack scenarios.

Moreover, because the IoT is only booming recently, it will also coexist with legacy systems that were not designed in such connected environment. This opens up new potential security breaches in well installed industries.

In addition to that, it is difficult to embrace and reply properly to all use cases that IoT will create. For example, many IoT devices will be deployed for years on the field, with a huge number of them not able to apply patches to update and level up their security features. Also, the IoT devices having constrained resources, it could not be feasible to apply the same resource demanding security mechanisms traditionally used in IT systems.

IoT devices are thus believed to be the weakest point of the IoT, and therefore, attract the majority of the attackers' efforts hoping to corrupt the ecosystem.

4.2 Motivating examples

The high tempo imposed by newer technologies, along with a long deployment time in sometimes hardly accessible areas, makes it hard to keep software systems up-to-date and secure. Two scenarios are here presented to illustrate this challenge within IoT devices.

4.2.1 Software-over-the-air updates

The software over-the-air (SOTA) update process is an important operation for maintaining a suitable level of software security. When applied to the IoT devices, it takes the name of SUIT (Software Updates for the Internet of Things). The purpose is to fix bugs, add new features and adapt the firmware to evolving environments. It allows software upgrades with feature enhancements, but also security patches to be distributed. The Internet Engineering Task Force (IETF) has a working group that emits recommendations for the SUIT process [29], still in draft status.

The thought architecture for the transport of a firmware update is described in the SUIT architecture document[27]. It involves notably the IoT device to update, a firmware image which is the binary containing the update, the manifest which is associated to a firmware image and contains meta-data, and a firmware server where the updates are stored and sent to by an author. A rough sketch based on an IoT robot scenario is presented in Figure 4.1. It shows the

possible interactions when the IoT firmware developer publishes a new update (firmware image associated to a manifest) on the firmware server and retrieved by the IoT device through the internet, possibly redirected by gateways on the way.

In the SUIT information model description[28], the group presents the threat model considered for the transport of firmware updates. Then, security requirements are derived to mitigate the exposed threats. They include:

- installation regulation through the use of sequence numbers, vendor and device-type identifiers, an update expiration time;
- authenticity of the firmware, the manifest, the payload type, the storage location, the remote resource location through full image digest and MAC (Message Authentication Code);
- secure execution through firmware verification at boot and compatibility checking by authenticated vendor and class IDs, the authenticity and digest of the payload size;
- rights granting through authenticity, access control;
- confidentiality through encrypted payload and manifest.

This process ensures that a firmware update is delivered using state-of-the-art security mechanisms and can't be reverse-engineered. It takes into consideration the low resource constraints of the devices and develops a scenario where the new firmware is retrieved from a firmware server, protected on the way from the author to the device and it assures that the replacing firmware is the one intended by the developer.

Other software update mechanisms have been proposed in the past and used nowadays, such as CHAINIAC [45] for decentralized software updates or TUF (The Update Framework) which is a general framework supporting compromised repositories or signing keys and all its variants like Uptane [37] specifically targeting secure SOTA vehicle updates.

4.2.1.1 Security issues and limitation of SUIT

While the update process is delivering the firmware in a secure manner, no functional guarantee is held on it. The behavior of the device could then change radically after an update and create security breaches, intentional or not. Also, malicious firmware could be placed on the firmware server and downloaded directly by the device, corrupting the device at the same time.

Another concern is that in such update, the firmware is composed by the applications but also other software units like an OS. How to check specifically the security of the applications?

Lastly, companies could choose not to disclose the source code of the applications they provide. How to conduct an analysis of IoT device application without any source code made available by a third-party provider ?

Hence, neither the threat model nor the security requirements presented above are able to cover the issue of a malicious/vulnerable software code bundled with the firmware update. These concerns are depicted in the following Figure 4.1 where the IoT device retrieves an update in a secure manner, however the flawed or malicious application made him out of order anyways.

An intuition to solve this security issue would be to analyse specifically the IoT application and infer the inherent security properties that hold in it, to be compared with the desired behaviour. From the moment when the firmware is sent to the server, the security mechanisms in place with SUIT prevent any modification or reverse-engineering capabilities. The IoT application analysis should thus be placed between the production of the firmware and its submission to the SUIT pipeline.

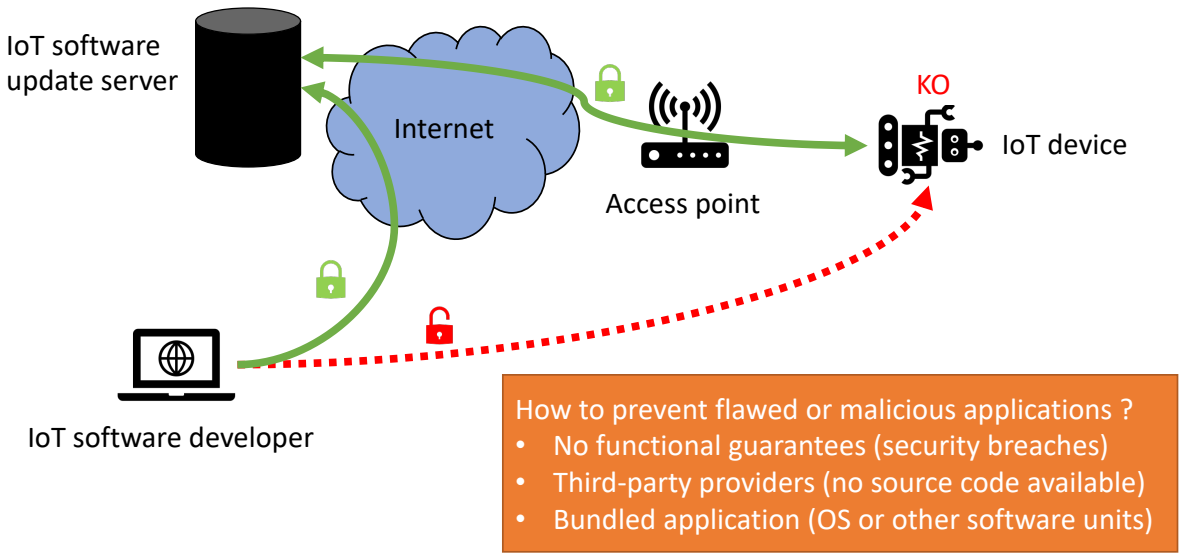


Figure 4.1 – A SUIT process scenario (inspired by [69]) with spotted security limitations

4.2.2 Computer-assisted security audit

In the IoT, the devices are expected to be deployed for years and will need permanent monitoring and regular updates. This could be done in the frame of a management engine that keeps track of the software and their version deployed in the field. If a vulnerability is identified, or an attack on going that threatens the ecosystem, the engine could exactly understand and anticipate the consequences. This opens up the possibility of a run-time risk assessment and real-time countermeasures.

However, how would it be possible to ensure a device's security properties at any time, for example given new discovered threats after the deployment of the device ?

4.3 Goal and objectives of the study

The apparent difficulty of properly assessing the cyber risks within the IoT ecosystem leaves users and companies in a permanent vulnerable state that the Research and the Industry are trying to compensate.

4.3.1 Related works

Due to the lack of standardization and usability of the exposed methodologies, researchers are calling for new approaches to support the expected high-paced IoT evolution.

Therefore, the future of IoT risk assessment is built around automated and continuous risk assessment in order to support the growth of the IoT and respond properly to incoming threats. This involves to adjust risk indicators in near real-time given internal and external factors within a continuous risk calculation engine and for that having at disposal support tools for systems' simulation and modeling [1].

However, workshops and interviews conducted with industry professionals and experts, reported in [47], showed uncertainty in the development of a fully automated risk assessment process because of the presence of social and human aspects in systems and due to the precise but inaccurate numeric approaches used in the assessments. It was believed that computer assisted assessment approaches had more opportunities to be valuable. This last assessment method means that a software could follow the IoT ecosystem and track the status of the deployed devices, in particular the software running on them and the patches applied.

Furthermore, there were discussions about collaborative risk assessment, where companies interact with partners, and need understanding of the shared risk. However, it is strongly thought that no company would openly share their own IoT risk data with partners without guarantees stated in a contract. Thus, the IoT systems would need to operate in an assumed state of compromise [47].

Moreover, lots of efforts are brought to the security of each layer of the IoT architecture (described previously in section 3.1.2), and are presented in [36], and with a particular focus to the application layer in [61]. Limited efforts deeply focus on IoT device applications.

Among them, SOTERIA [11] is a framework, based on static analysis, that automatically extracts a model out of an IoT application and use a model checker to validate application specific properties. It first extracts an intermediate representation (IR) from the source code, from which a state model of the application is generated, eventually confronted to a set of IoT properties for a standalone or interacting applications.

IoTGuard [12] enforces safety and security policies by adding instrumentation code to an application's source code. The instrumented code blocks undesired states when they correspond to policy violation. The framework creates a model of individual applications or interacting ones.

IoTSan [43] also relies on the source code to translate an application into a model that will feed a model checker. It considers applications, devices and their configurations.

In the mobile security field, BYODroid [4] is a security policy checker for android devices, based on model checking. The framework verifies security policies of android application from their executable. It also proposes a meta-market where only applications that are compliant with a defined security policy are made available.

In that perspective, our work also relies on model checking based on static analysis. It targets IoT device application but, in contrast to the other IoT frameworks, exclusively relies on the executable of an IoT application.

4.3.2 Problem formulation

As seen previously, the current research showed it seemed promising to develop and explore this orientation of IoT device application security. This comes in the broader perspective of securing the IoT ecosystem.

The adopted strategy in this discussion is to secure the application layer of an IoT device. If such goal is reachable, the application would participate in reaching a trusted unit within the IoT ecosystem by the support of a secure OS. The target goal is the development of an automatic security software audit tool working as a security policy-checker.

Firstly, the tool must be directed to IoT devices that don't even have enough resources to guarantee their security by their own, thus putting at risk the whole ecosystem. As such, the proposed tool would target low-end IoT devices and look for inherent security properties.

Secondly, because of the diversity of IoT devices, the tool should target different architectures such as the dominant ARM and MIPS.

Finally, it must be able to analyse applications that are already deployed or about to be deployed, so with limited or no access to the source code. Thus, the security properties to check will lie in the executables.

Chapter 5

Proposed solution : A framework for the automatic security analysis of IoT device applications

5.1 The framework

In the exposed motivating example of the SUIT process, we don't want the new firmware to behave in a non-secure manner in the ecosystem and thus we would like to understand its behaviour from a security standpoint before field deployment. In particular, we would like to verify that the system behaves without threatening its security. The security constraints imposed on the system can be expressed in terms of policies, which are formal specifications regulating the behaviour of a system. In other words, the goal is to identify the security properties of a system w.r.t. a desired policy.

5.1.1 Overview

Another technique than testing or simulating a model in order to verify the intended behaviour of a system is to do a formal verification. It requires a model of the system, that is to say an abstract representation of the real system.

One approach of formal verification is model checking [50], which verifies if a given specification holds in a model. A major benefit in using model checking is the often fully-automated process to exhaustively explore the model. Hence, we propose model checking as the approach to form the foundations of our framework to automatically analyze the IoT device applications.

In addition to that, we would like to investigate the security properties of the application before its deployment to field devices. For this reason, we prefer a static analysis approach. This is to be compared to dynamic analysis of code, where the program needs to be executed. This approach doesn't imply a full test coverage of the code, which wouldn't be acceptable in a security checking.

Furthermore, the analysis should be conducted even with no source code available, thus relying on the executable only. Many reasons support this choice, explained earlier in 3.2.3.7.

The workflow imagined to satisfy the expressed needs is depicted in Figure 5.1. It starts by generating the model of an IoT device application and by specifying the security policy it should respect. At the end are given compliance results which state if the security properties described in the policy are satisfied or not. Each component is described in details in the next subsections.

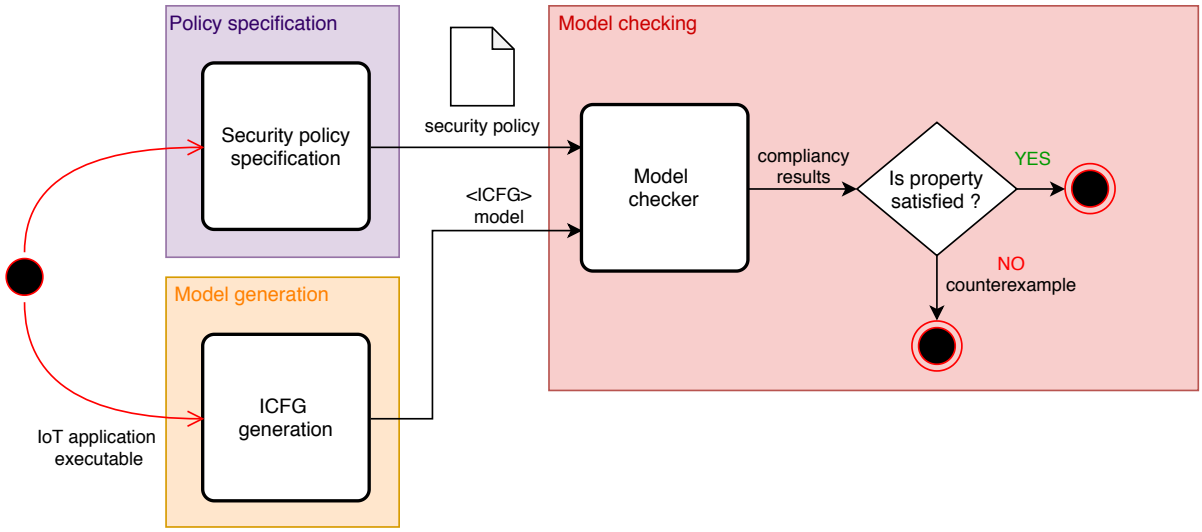


Figure 5.1 – The framework for automatic analysis of IoT device applications

5.1.2 Security model generation

Security properties from the executable could be exported out by conducting a program analysis. Indeed, as exposed in 3.2.3, it is possible to model a program by computing its ICFG (interprocedural control-flow graph) for example.

Since we aim to describe the behaviour of a system from the security perspective, any operation not relevant from the security standpoint don't participate in the security model. For this, we list all operations that would need security considerations, *viz.* file operations, cryptographic primitives and network procedures. All of them constitute relevant APIs to our model, any other operation are tagged irrelevant. In such way, we track the relevant APIs in the model while the irrelevant ones are left aside.

The retrieved model exposes the expected behavior of the program : a simplified, abstracted version of the more complex reality of the system. It gives us the opportunity to directly infer some properties.

The security model of an application is the desired asset to conduct a model checking of properties described in the security policy.

5.1.3 Security policy specification

A policy describes the properties that should hold in the model, while properties mirror a system description that can be formally expressed. For systems, the security policy is a set of requirements to keep a system protected from threats, according to an entity. It groups the authorized and unauthorized properties and behaviors of a system.

In our SUIT motivating example, we maybe would like to express the policy that file operations are not allowed. The associated property to verify would be that no function enabling a file operation is used. Using a Kripke structure, it would be to check that no state are making use of such function.

These properties are expressed in a formal language, often using temporal logic in a Kripke structure. To each property to be verified is assigned a temporal formula to be validated. Temporal logic has many types, such as Linear Temporal Logic and Computational Tree Logic that we will explore next.

5.1.3.1 Linear Temporal Logic

In the Linear Temporal Logic (LTL), there is only one possible future, so at each moment of time, there is only one successor state. This forms a sequence of states, thus a linear notion of time. It describes the events happening along a time path.

One can express properties about the sequence of states with temporal operators, some of them are given in the following list. From a given state,

- $\mathbf{X}p$ ("next"): property p holds in next state;
- $\mathbf{F}p$ ("eventually"): property p will eventually hold at some point in the future;
- $\mathbf{G}p$ ("globally"): property p always holds in the future;
- $p\mathbf{U}q$ ("until) : property p holds at least until property q holds (which will hold in the future).

5.1.3.2 Computational Tree Logic

The LTL represents a single computation of the system, with only one possible future. Instead, the Computation Tree Logic (CTL) is based on a branching notion of time. This means for each moment of time, there can be several possible futures. Each state can thus have several successors, giving a tree of states. And each path in the tree represents a single computation of the system.

In order to express if a property holds for all paths or some of them, two quantifier operators are introduced: the **A** operator ("for all paths") and the **E** operator ("there exists a path").

For example, the formula "$\mathbf{AG}p$" states that the property p should hold at each state of any path, whereas the formula "$\mathbf{EG}p$" states that there exists a path where the property p always holds (and eventually some paths that never hold property p).

5.1.4 Model checking

The properties that are relevant in the security context are kept from the extracted model and confronted to the security policy. The outcomes are the compliance results: the properties are verified or not.

5.1.4.1 Compliancy results

With a model represented as a Kripke structure, it is thus possible to verify some properties at any time (or state of the system). The model checking process ends with compliancy results, so if the properties of the policy were satisfied or not given the model, and thus in the IoT application.

In case of a failure during any stage of the process, also if the security policy is not respected, the check is considered negative. This implies it could wrongly mark applications that meet the security policy as false positive.

5.1.4.2 The state explosion problem

One of the most important issues in model checking is the so called *state explosion problem*[14]. In order to check some properties, the model checker needs to explore the entire state space, which complexifies as the number of states grows large. Our security model addresses this issue because, as explained earlier, any operation with no security relevant APIs are tagged as irrelevant. This means we will have paths of calling procedures with no significance from the security standpoint. All these paths could be pruned from the retrieved ICFG without interfering with the soudness or the completeness of the model. With all these paths and associated

states discarded from the model, the model checker will face a model with less complexity and thus suffer less from the *state explosion problem*.

5.2 Implementation of the framework

The target goal is thus to extract the security model of the application and to confront it with user-defined desired security properties. Given the presented framework, we used different support tools and developed our own custom model simplification and policies.

5.2.1 Overview

The proposed framework has been implemented using different support tools, described in the following section and visualised in Figure 5.2.

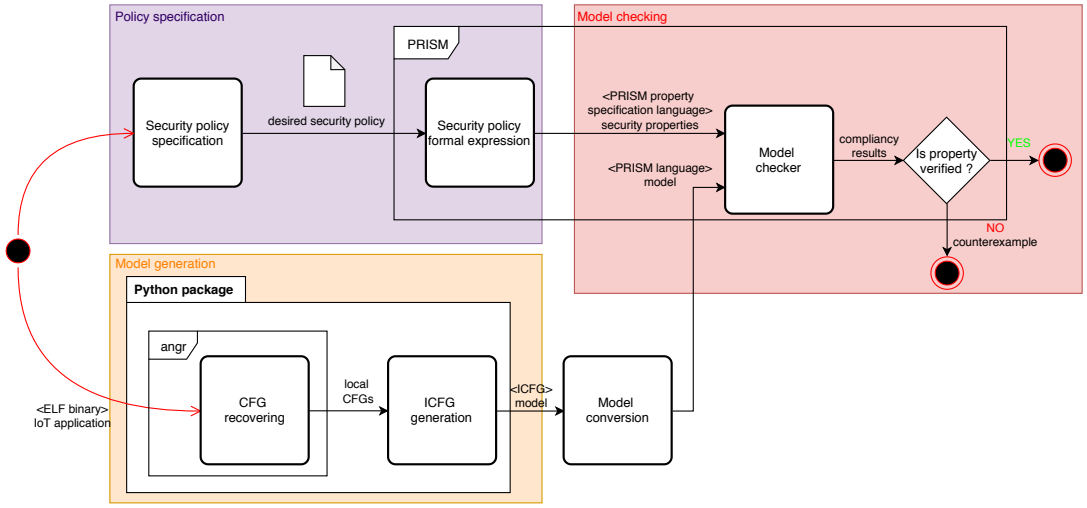


Figure 5.2 – The implemented workflow of the security framework

5.2.2 Security model extraction : the angr framework

Angr[62, 59] is a python-based binary analysis framework used for program analyses developed by the Computer Security Lab at UC Santa Barbara. It is a popular open-source binary analysis tools, world-wide known for being used during the DARPA Cyber Grand Challenge where teams using it reached the top ranking. Another very popular tool that we could have used is the BAP (Binary Analysis Platform) [15], with differences highlighted in [10].

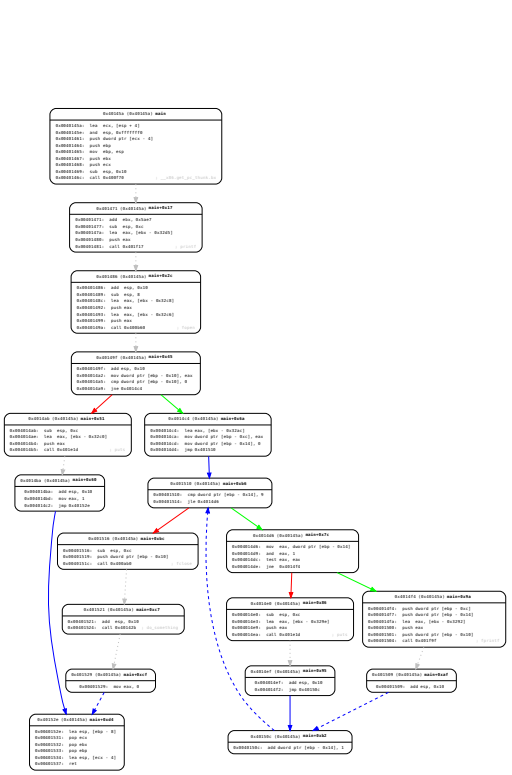
Angr allows easy expansion to write both static and dynamic symbolic analyses by providing tools to directly interact with the binary: solver engine, machine-state emulation, program path analysis, semantic representation and full program analysis. It currently supports the most common architectures including x86, ARM, MIPS and AMD.

Angr disposes of modules to visualize and explore the CFG resulting from the analysis of the application. As seen in 3.2.3, the CFG represents the behavioral model and fits our needs for properties inference.

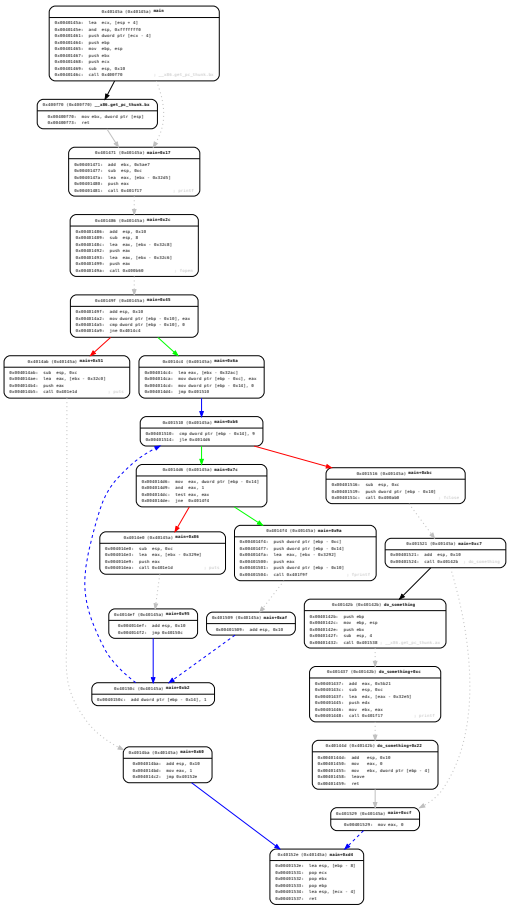
Each CFG in angr has its own sets of nodes and edges. A node is constituted by a basic block (c.f. section 3.2.3.2), and can be retrieved by its address using angr's Function Manager. Each edge links two nodes with the kind of jump performed between them (a call, a system call, a return or a plain transition).

While the analysis covers the full executable, the retrieved CFG in angr depends on the selected entry point and doesn't spread to the callees. Thus, it is needed to post-process the

CFG in order to create the ICFG (c.f. 3.2.3.2) of the program by binding the main function's CFG with its callees' CFG. To achieve this, we developed an algorithm that explores the graph from the main function's CFG, removes angr's *FakeRet* edges (part of angr's own processing which guarantees that a callee returns) and creates clone nodes for callees that have already been inserted in the graph (to avoid duplicate nodes if their return address is not the same). The effect of this algorithm can be seen visually in figure 5.3.



(a) CFG graph visualisation without the callees (pure CFG)



(b) CFG graph visualisation with the callees (ICFG)

Figure 5.3 – Comparison of a CFG with and without the callees included

At this point, the complete ICFG, composed by the main function's local CFG connected to its callees' CFG, describes the full control flow of the application. This represents the behavioral model of the application.

5.2.3 Security policy specification: the PRISM language

Many model checkers exist, holding different properties and using different model languages. *PRISM* (Probabilistic Model Checker) [38] is one of them, is free, open-source, and has a tool to analyse systems according to probabilistic behaviours. It also

In PRISM, policies are sets of states of the model we want to check. They are represented in logical expressions. The tool supports model-checking of non-probabilistic properties using LTL and CTL, which suits our needs for the policies we would like to express. Indeed, the properties we would like to check with PRISM are the use of APIs (Application Programming Interfaces) and the call ordering in all execution paths. We particularly focus on APIs with a security meaning, because they are the only ones relevant in a security policy, *viz.* any file operation,

crypto primitives. Each security property is thus expressed in terms of a logical expression to be evaluated by the tool, as in the following example.

```
// No file read operation using fread (FRD)
"never_fread" : A [G FRD=0];
```

Here, the example presents the following policy expressed in the PRISM property specification language: ban any use of the *fread* C function. It states that along any path from the initial state, no state should set the *FRD* variable. In other words, the variable representing the *fread* function should never be present in the model.

PRISM can then compute the probability of the logical expression to be true. For a security property, it means PRISM can say whether or not this property holds in a model.

With non-probabilistic expressions, PRISM can also generate counterexamples and witnesses for further investigation on a failed property verification. This allows human investigation as a post-process to determine the reason(s) of the policy check failure.

Hence, in our use case, security policies could be translated into PRISM expressions using LTL and CTL. PRISM can then confront these policies to the application's model retrieved earlier.

5.2.4 Model checking: model conversion and PRISM model checker

5.2.4.1 Model conversion: from ICFG to a PRISM model

However, the raw ICFG, from the model extraction phase, is not recognised by PRISM, which should be fed with a model written in the PRISM language. Thus, the retrieved ICFG should be converted into this language to be used. Furthermore, the extracted model in the ICFG shape could be heavy to interact with. Thus, for performance purposes and depending on the size of the application, a PRISM model would be easier to use.

However, the direct conversion from ICFG to PRISM language, while feasible, is quite difficult. Thus, the model is temporary converted in the MCRL2 language before being converted in the required PRISM language. This is encouraged by the more simple representation of a model in the MCRL2 language and the fact we can rely on tools to automatically convert from the MCRL2 language to the PRISM language thanks to shared characteristics in the model representation, as shown in table 5.1.

MCRL2 language	PRISM language
• State-based composed of processes • Each process can carry data; – the specific combination of data defines the process' state. • The processes can perform actions – combined actions through algebraic operators (.,+) form new processes; • Each process has a corresponding state space – contains all the states a process can reach; – contains all the possible transitions between these states. • Very complex systems, e.g. containing a lot of parallelism, can be translated into a single linear process.	• State-based composed of modules – modules interact with each other. • Each module has internal variables – the values of these variables at any time constitute the state of the module. • Each module describes a behavior – a behavior is the transition between each local state when a predicate is met; – each update will be assigned a probability of execution. • The global state of the model is determined by the local state of each module.

Table 5.1 – Comparison of the MCRL2 and PRISM languages

5.2.4.2 The MCRL2 language

A MCRL2 model is a plain-text file written in the MCRL2 language. It describes processes, all together describing the behavior of the overall system. Each process can perform actions that are any events that can happen in the system (read, write, send mail...). The combinations of actions and processes form new processes and are done through algebraic operators. For example, the "+" operator mimics an alternative behavior to represent a choice between two behaviors, whereas the "." operator is the sequential composition to represent sequences of actions.

```
% MCRL2 MODEL OF ELF FILE: thesis_example_no_hash.elf
act skip;
%<printf>
PRNT;
%<fopen>
FOPN;
%<fclose>
FCLS;
%<fprintf>
FPRNT;
%<puts>
PTS;
%<putc>
PTC;
...
proc
EXTAPI=skip;
%<CFGNode __x86.get_pc_thunk.bx [4]>
```

```
__x86.get_pc_thunk.bx = (skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip
    +skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+
    skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+
    skip+skip+skip+skip+skip+skip+skip+skip);
%<CFGNode __x86.get_pc_thunk.ax [4]>
__x86.get_pc_thunk.ax = skip;
%<CFGNode do_something [12]>
do_something = __x86.get_pc_thunk.ax.PRNT.skip;
%<CFGNode main:0xd4 [10]>
main:0xd4 = skip;
%<CFGNode main:0xbc [11]>
main:0xbc = FCLS.do_something.main:0xd4;
%<CFGNode main:0xb6 [6]>
main:0xb6 = (main:0xbc+main:0x7c);
%<CFGNode main:0xb2 [4]>
main:0xb2 = main:0xb6;
%<CFGNode main:0x9a [21]>
main:0x9a = FPRNT.main:0xb2;
%<CFGNode main:0x86 [15]>
main:0x86 = PTS.main:0xb2;
%<CFGNode main:0x7c [10]>
main:0x7c = (main:0x9a+main:0x86);
%<CFGNode main:0x6a [18]>
main:0x6a = main:0xb6;
%<CFGNode main:0x51 [15]>
main:0x51 = PTS.main:0xd4;

init
hide({skip},__x86.get_pc_thunk.bx.PRNT.FOPN.(main:0x6a+main:0x51));
```

MCRL2 model of the example application in 6.2 without hashing the identifiers

The ICFG of the application is made of nodes which have maximum 2 out branches, each edge representing a jump in the flow. Thus, the conversion of the directed graph to a MCRL2 model is straightforward: the nodes represent the processes in the model, the called APIs represent the actions, and the behavior of the system is described by the edges. The edge representation depends on the number of child nodes: when maximum one edge is connecting two nodes, they should be combined sequentially (“.” operator); when two edges leave a node, they should be combined alternatively (“+” operator).

In the above MCRL2 model example, we can distinguish the different sections of a MCRL2 model: the keyword *act* introduces the actions, the keyword *proc* introduces the processes and the keyword *init* is used to indicate the start process(es).

The capitalized functions are used in the security policy definition in order to refer to the relevant security APIs.

5.2.4.3 Hash matching

The last step is to make the MCRL2 model compatible with the PRISM model. Indeed, the PRISM language accepts only identifiers, i.e. names given to variables and modules, which correspond to the following regular expression : [A-Za-z_][A-Za-z0-9_]*. All identifiers in the MCRL2 model are thus hashed and a capital letter added at the front, except for the relevant security APIs that need a stable denomination (see section 6.2).

5.2.5 The security policy-checker tool

The implemented framework connects an IoT device executable to a model checker. All components have been wrapped up in a single tool and bash and python scripts support the tool to fully automate the policy-checking procedure. It has been integrated in the Git code version control system. The automatic security policy-checker tool is represented in figure 1.1.

5.2.6 Enhancements of the policy-checker tool

Some enhancements have been brought to the policy-checker in order to accelerate the model checking.

In a computer program, some functions can be called several times and thus appear the same number of times in the ICFG. Thus, these functions could be modeled only once and referenced in the rest of the ICFG. The algorithm is in charge of pruning the ICFG tree when detecting already modeled functions and adds the proper references instead.

For this policy checking, we were only interested in the security model of the application. All non relevant security APIs are thus skipped, as well as all nodes that don't contain a callee (a simple program jump for instance). The keyword *skip* is thus introduced to bypass these irrelevant operations and accelerate the modeling, as can be seen in 5.2.4.2.

We also included a visual representation of the final ICFG, once all the callees have been retrieved and merged, for verification purposes and further exploitation like graph analysis. This allows us to check the soundness of the model. For that, we used the bingraphvis framework [7] (see given example 5.3) .

Chapter 6

Experiments

6.1 Experimental framework

Our experiment has been conducted on a Packard bell EasyNote NM with Ubuntu 18.04.2 LTS, Intel Pentium (R) P6200 @2.13GHz * 2, 4 GB DDR3 Memory and 50GB HDD.

6.1.1 Secure the application layer of low-end IoT devices

IoT devices form a disparate landscape of different architectures, performances and operating systems (OSes). Several projects are gaining attention and focusing efforts to impose their solutions to the development of the firmware running on the IoT devices. However, each project targets specific system configuration, depending on the desired features and the performances goals. For example, high-end IoT devices require more resources since they are designed for advanced features and provide multiple technologies, whereas low-end IoT devices are designed to be minimalistic (see 3.1.7).

Security should cover the entire landscape, especially low-end devices which have limited resources to defend themselves against attacks. Thus, the selected use case is centered around low-end devices, being thought as the weakest part of the IoT ecosystem from a security standpoint.

Among the major players in the open-source IoT OSes for low-end devices, RIOT-OS [8] draws attention for its active development and its ambition to be the favorite OS of IoT applications developer [18]. Thus, RIOT-OS has been selected to be the OS on which applications we want to secure are running on top.

6.1.2 RIOT-OS: The Revolutionary Internet of Things – Operating System

RIOT-OS [8] is a soft real-time micro-kernel inspired by Linux but built from scratch. It specifically targets IoT low-end devices that need a very minimal resource usage (RAM, ROM, CPU, power consumption). Most third-party libraries are avoided and rewritten and RIOT-OS provides a hardware abstraction to be as portable as possible, allowing application code to be reused on a different hardware. The applications can be developed with the programming languages C and C++.

Furthermore, common IoT OSes are often closed-source with vendor-locked software. RIOT-OS, on the contrary, is open-source and provides a more complete understanding of the application's behavior by having a view on its inner working. This revealed to be very beneficial for developing applications, but also testing and verifying the developed tool.

In addition to that, the OS is built in a modular manner, only the needed modules are loaded and compiled to reduce the consumption of the resources. They provide features to the applications, like connectivity or timers.

The applications and the OS are compiled and linked together in an *Executable and Link Format* (ELF) file. The created file can be executed directly on the machine corresponding to the targeted architecture. Furthermore, the compiled ELF file of the RIOT application is valid for conducting static analysis with the angr framework.

RIOT provides an abstracted board, called *native*, that can emulate a hardware when run as a process on a *NIX system. Only the RIOT software stack is packed, dropping all hardware specific commands. This helps developers to abstract from a specific hardware, to assist the development without limitations of debugging on the hardware, and conducting experiments before the hardware is available.

6.2 Proof of Concept

When the IoT device is initialized, RIOT-OS launches two threads: the *idle* thread, which has the lowest execution priority and puts the system into low power mode ; and the *main* thread, which is the user application. Indeed, in low-end devices, usually only one program runs on the target device, which is how it works in RIOT-OS. In this section, we will develop an example program, described in the following C file *main.c*.

```c
#include <stdio.h>

int do_something(void)
{
    // Do something....
    printf("I do something!");
    return 0;
}


int main(void)
{
    printf("Hello World!");

    FILE *f = fopen("log.d", "w");
    if (f == NULL)
    {
        printf("Error opening file!\n");
        return 1;
    }

    const char *text = "Text to write";

    for(int j=0 ; j < 10 ; j++){

      if(j%2==0){
        puts("No writing\n");
      }
      else {

        fprintf(f, "%d: %s\n", j, text);
      }
    }

    fclose(f);

    do_something();
    return 0;
}
```

example/main.c

6.2.1 Program explanation

The program starts with the main function, and first outputs the initial message "Hello World!" (line 13). Then it tries to open a logger file (line 15), called *log.d*, in writing mode, and if it can't, ends the program (line 19).

If the file opens, it enters a 10-iteration loop (line 24), where it will alternatively display the message "No writing" (line 27) and write the number of the iteration with the message "Text to write" in the logger file (line 31). The programs ends by closing the logger file (line 35) and calling the *do_ something* function (line 37) that just prints out the message "I do something!" (line 6).

The outputs of the program are:

```
1  RIOT native interrupts/signals initialized.
2  LED_RED_OFF
3  LED_GREEN_ON
4  RIOT native board initialized.
5  RIOT native hardware initialization complete.
6
7  main(): This is RIOT! (Version: 2019.07-devel-507-gbf1b3-HEAD)
8  Hello World!
9  No writing
10
11 No writing
12
13 No writing
14
15 No writing
16
17 No writing
18
19 I do something!
```

6.2.1 : Console outputs when running the example program with the *native* board

The logger file *log.d* has been created and contains the following text:

```
1: Text to write
3: Text to write
5: Text to write
7: Text to write
9: Text to write
```

6.2.1 : Outputs written in the logger file *log.d*

This example is not a proper IoT device application since no remote interaction is possible. However, adding the connectivity would create too many satellite explanations that are not necessary for a first insight. Later, in 6.3, we expose the results with sample IoT device applications.

6.2.2 Crossing the framework

From this program, angr can retrieve the base address of the main thread by looking for the *main* symbol and compute the local CFG of the function. Indeed, a RIOT application will always start by a call to the *main* function, so our custom algorithm that constructs the MCRL2 model will take the CFG's first node as the start process, and expand from there. Each callee will then in turn be considered as new processes, called by a previous process. The MCRL2 model of the example applications is presented in 6.1.

```
1  % MCRL2 MODEL OF ELF FILE: thesis_example.elf
2  act skip;
3  %<printf>
```

```
PRNT;
%<fopen>
FOPN;
%<fclose>
FCLS;
%<fprintf>
FPRNT;
%<puts>
PTS;
...
EXTAPI=skip;

%<CFGNode __x86.get_pc_thunk.bx [4]>
H4405035592032427656 = (skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+
    skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+
    skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+
    skip+skip+skip+skip+skip+skip+skip+skip+skip);
%<CFGNode __x86.get_pc_thunk.ax [4]>
H4399155066186473056 = skip;
%<CFGNode do_something [12]>
H4397726874126121931 = H4399155066186473056.PRNT.skip;
%<CFGNode main+0xd4 [10]>
H4399169596965014106 = skip;
%<CFGNode main+0xbc [11]>
H4399195794892733106 = FCLS.H4397726874126121931.H4399169596965014106;
%<CFGNode main+0xb6 [6]>
H4399189273326958056 = (H4399195794892733106+H4397680298621147506);
%<CFGNode main+0xb2 [4]>
H4399210018840388156 = H4399189273326958056;
%<CFGNode main+0x9a [21]>
H4397642149099981556 = FPRNT.H4399210018840388156;
%<CFGNode main+0x86 [15]>
H4397665484065172856 = PTS.H4399210018840388156;
%<CFGNode main+0x7c [10]>
H4397680298621147506 = (H4397642149099981556+H4397665484065172856);
%<CFGNode main+0x6a [18]>
H4399011950964517956 = H4399189273326958056;
%<CFGNode main+0x51 [15]>
H4399036995417465931 = PTS.H4399169596965014106;

init
hide({skip},H4405035592032427656.PRNT.FOPN.(H4399011950964517956+
    H4399036995417465931));
```

Listing 6.1 – MCRL2 model of the example program

This MCRL2 is then converted in a PRISM model to feed the PRISM model checker.

The soundness of the MCRL2 model can be verified by our custom visualization tool, represented in figure 6.1. The complete structure of the program can be perceived in the figure, with the ascending dashed blue line representing the loop, the green and red arrows separating the if-else condition cases of the alternating message displaying and text writing, and in the bottom left hand corner, the final file closing and the *do_something* function call. This structure is to be compared with the MCRL2 model, where the *printf*, *puts* and *fprintf* functions are the only relevant security APIs to monitor, so the only ones that are not hashed.

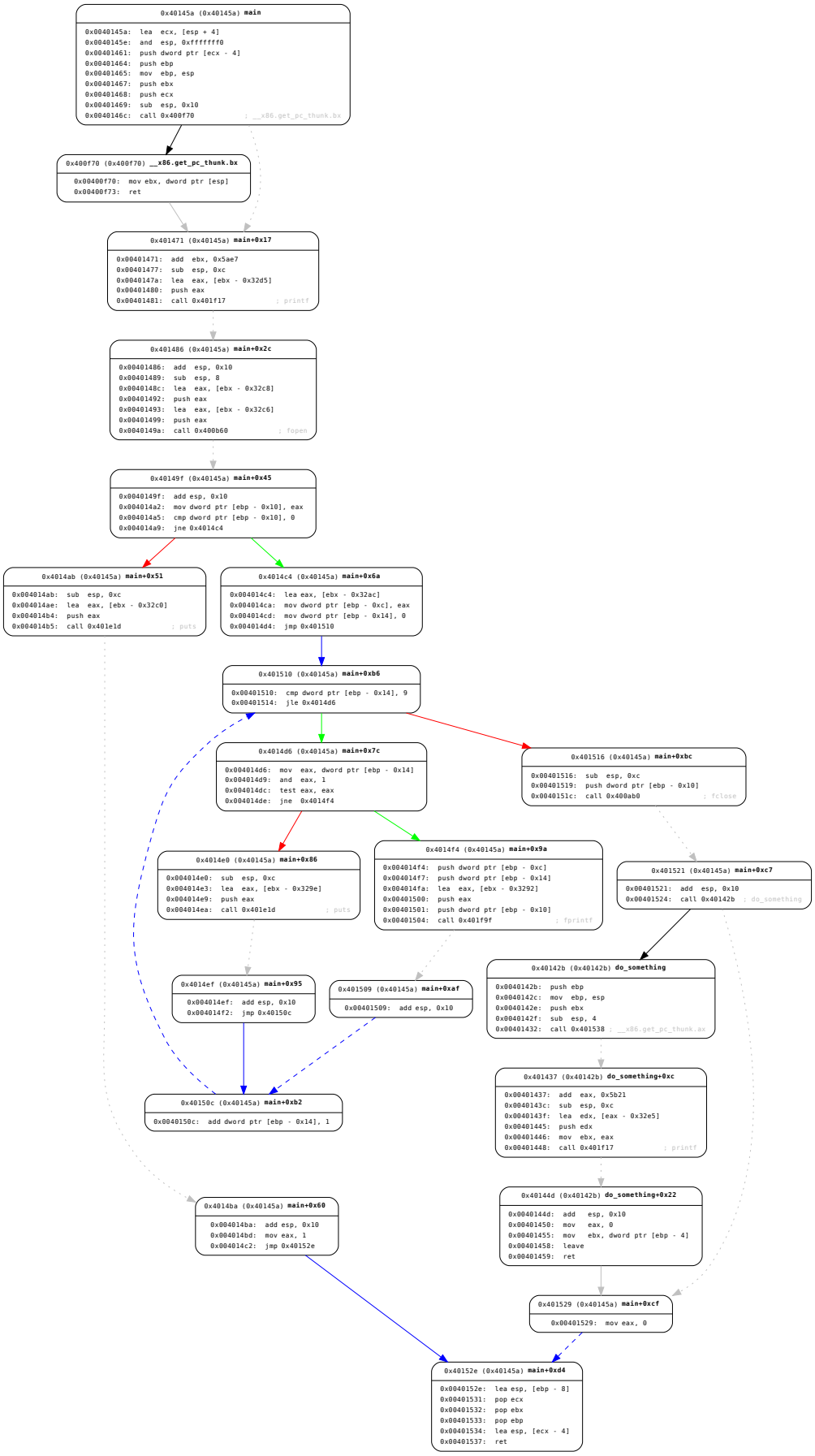


Figure 6.1 – Full CFG of the program

The security policy for this example is represented in the following bloc:

```
//Unsecure primitives : Use of deprecated or unsecured libraries (crypto primitives, connections...)
// List :
// (https://stackoverflow.com/questions/2565727/which-functions-from-the-standard-library-must-should-be-avoided & https://stackoverflow.com/questions/26558197/unsafe-c-functions-and-the-replacement) : strcpy, strncpy, strcat, strncat, sprintf, vsprintf, gets, makepath, _splitpath, scanf/sscanf, snscanf, strlen, atoi, atof, atol
"no_unsafe_C_primitives" : A [G SCPY=0 & SNCPY=0 & SCT=0 & SNCT=0 & SPRNT=0 & VSPRNT=0 & GTS=0 & MKPTH=0 & SPTH=0 & SCF=0 & SSCF=0 & SNSCF=0 & STLN=0 & ATI=0 & ATF=0 & ATL=0];

// File operations
// No data local access : No read from file
// 1 - No data local access : No read from file with fread
"never_fread" : A [G FRD=0];
// 2 - No data local access : No read from file with fgets
"never_fgetc" : A [G FGTS=0];
// 3 - No data local access : No read from file with fgetc
"never_fgets" : A [G FGTC=0];

//No data persistency : No data stored locally on the device
// 1 - No data persistency : No write in file with fprintf
"never_fprintf" : A [G FPRNT=0];
// 2 - No data persistency : No write in file with fwrite
"never_fwrite" : A [G FWRT=0];
// 3 - No data persistency : No write in file with fputs
"never_fputs" : A [G FPTS=0];
// 4 - No data persistency : No write in file with fputc
"never_fputc" : A [G FPTC=0];

// Bad programming
// 1 - Never use file operations if no fopen active
"always_fopen_if_fops" : A [G (FCLS=1 | FRD=1 | FGTS=1 | FGTC=1 | FPRNT=1 | FWRT=1 | FPTS=1 | FPTC=1) => FOPN=1];
```

example/example_policies.props

The first checked property is that no deprecated or unrecommended functions should be present in the code, identified by the label *"no_unsafe_C_primitives"*. This answers the OWASP top vulnerability number 5 for the "use of insecure or outdated components" (c.f. appendix B). When extracting the properties within the model (the relevant APIs), this expression checks that for all paths from the initial state, at no point the APIs exposed in the expression are set, i.e. no insecure function is found in the flow of the program. This is formed by the use of the A and G operators.

The second set of property that is checked ("*No data local access*") forbids the use of the C functions that permit file readings, using the same expression structure as the previous property check.

The third set of property that is checked ("*No data persistency*") relates to the OWASP top vulnerability number 7 of "insecure data transfer and storage" (c.f. appendix B), further strengthened here as the restriction to not persist data locally. Similarly to the previous explanations, the prohibition of using C functions to store data is verified through the use of the A and G operators.

The last property ("*Bad programming*") performs the check that for all states of any path which involves a file operation, the file should have been opened earlier (*FOPN=1*). This prevents an undesired behaviour of the program at run-time.

6.2.3 Results

The PRISM checker verifies the compliance of the properties from the extracted model with the security policy, and render the results in the file *example/prism_thesis_example.result*.

```
"no_unsafe_C_primitives": A [ G SCPY=0&SNCPY=0&SCT=0&SNCT=0&SPRNT=0&VSPRNT=0&
    GTS=0&MKPTH=0&SPTH=0&SCF=0&SSCF=0&SNSCF=0&STLN=0&ATI=0&ATF=0&ATL=0 ]:
Result
true

"never_fread": A [ G FRD=0 ]:
Result
true

"never_fgetc": A [ G FGTS=0 ]:
Result
true

"never_fgets": A [ G FGTC=0 ]:
Result
true

"never_fprintf": A [ G FPRNT=0 ]:
Result
false

"never_fwrite": A [ G FWRT=0 ]:
Result
true

"never_fputs": A [ G FPTS=0 ]:
Result
true

"never_fputc": A [ G FPTC=0 ]:
Result
true

"always_fopen_if_fops": A [ G (FCLS=1|FRD=1|FGTS=1|FGTC=1|FPRNT=1|FWRT=1|FPTS
    =1|FPTC=1)=>FOPN=1 ]:
Result
true
```

example/prism_thesis_example.result

The result states that the program isn't compliant with the defined security policy, because of the failed check for the policy *"never_fprintf"*. It detected the use of the *fprintf* function, that allows data to be stored locally on the device, and thus failed the verification.

The complete verification took 4 seconds without the rendering of the CFG graphs, and 5 seconds including the CFG graph rendering time.

The same procedure has been conducted on the program compiled for the *nucleo-f030r8* board, based on the ARM Cortex-M processor core, which gave the same results. This validates the transfer of our approach to embedded device programs. Encouraged by the expected results, we decided to proceed with experiments on real IoT example applications to be sure of the applicability of the methodology for the IoT domain.

6.3 Experiments on IoT example applications

The experiment with the same settings as the dummy example has been reconducted on 21 of the applications in the example folder of the RIOT-OS repository [55], compiled for the *native*

board.

6.3.1 Results

The policy checker used the same security policy as proposed in the previous example. Nineteen applications failed to comply to this policy, because of failing the property about the use of deprecated C primitives (*"no_ unsafe_ C_ primitives"*). The failed message of one of these checks is exposed in listing 6.2.

```
"no_unsafe_C_primitives": A [ G SCPY=0&SNCPY=0&SCT=0&SNCT=0&SPRNT=0&VSPRNT=0&
    GTS=0&MKPTH=0&SPTH=0&SCF=0&SSCF=0&SNSCF=0&STLN=0&ATI=0&ATF=0&ATL=0 ]:
Result
false
```

Listing 6.2 – Fail message to the *"no_ unsafe_ C_ primitives" policy*

Applications	**Size (kB)**	**Time without graph rendering (s)**	**Time with graph rendering (s)**
asymcute_mqttsn	249.9	25	312
bindist	33.9	5	5
ccn-lite-relay	335.2	23	X
cord_ep	255	22	56
cord_epsim	195.9	30	X
default	151.1	12	38
emcute_mqttsn	235.8	23	118
filesystem	107.1	10	X
gcoap_example	249.9	23	47
gnrc_minimal	157.6	18	53
gnrc_networking	282.3	30	57
gnrc_tftp_example	286.6	30	55
hello_world	33.9	5	5
ipc_pingpong	38.4	6	104
lua_basic	335.3	74	X
lua_repl	339.6	243	X
nanocoap_server	184.1	32	X
ndn_ping	181.5	12	22
posix_sockets_example	240.3	20	46
saul_example	49.3	9	17
timer_periodic_wakeup	43.8	7	X
Total	3986.5	659	X
Mean	189.8	31.38	X
Standard deviation	105.1	50.79	X
Median	195.9	22	X

Table 6.1 – Statistics on the time taken by the tool to check 21 RIOT applications, with and without graphs rendering

Note: The undefined value "X" represents the time limit of 360 seconds, over which the rendering is aborted.

The table 6.1, graphically represented in figure 6.2, shows some statistics about the use of our tool on the example applications. This highlights that the bigger the firmware is, the more time is needed to verify it. However, it is not proportional since firmware of the same size won't perform exactly the same. This can be seen with *ccn-lite-relay*, *lua_ basic* and *lua_ repl*,

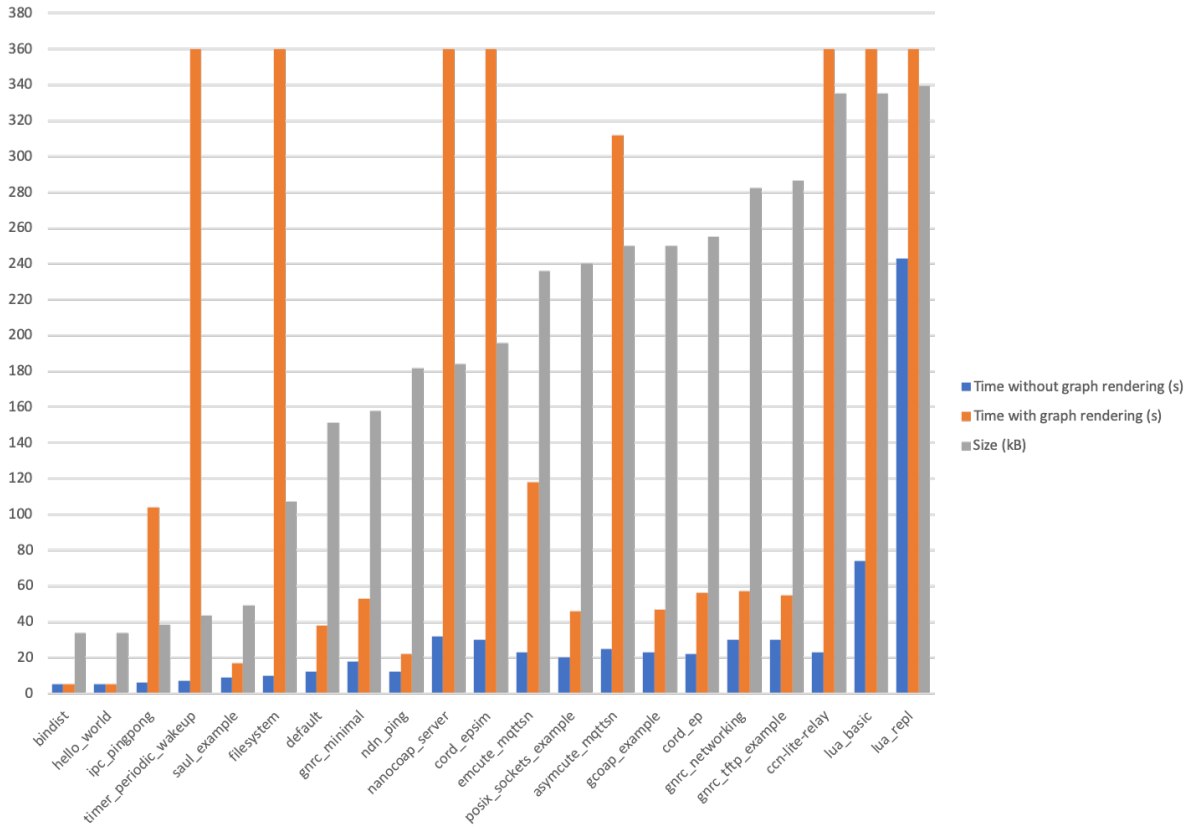


Figure 6.2 – Graph representing the statistics in 6.1

Note: The undefined value "X" has been replaced by the value 360, which was the threshold for the graph rendering before aborting the operation.

each with a size of around 337 kBytes, but the first completes the check without rendering in 23 seconds, the second in 74 seconds, and the last in 243 seconds (see figure 6.2). It took 659 seconds to check all the applications without rendering the graphs, so an average of 31.38 seconds per application check.

In any case, there is no correlation between the size of the firmware and the verification time including the graphs rendering. There are cases where the graph rendering period could not finish within the time limit of 360 seconds (6 minutes), mentioned as the value "X" in the table.

6.3.2 Verification

In cases where the graph rendering eventually ended, we checked manually the soundness and the correctness of the model and the validity of the results. This post-processing stage was necessary to achieve trust in the developed tool. The obtained results gave us sufficient positive results and enthusiasm to test the policy checker against a real-world scenario.

6.4 Experiment on a real-world scenario

This section presents a real-world scenario evaluation from an original work aimed to be published soon in a paper.

6.4.1 Use Case

6.4.1.1 Security Policy

In this use case, we will focus on the OWASP top 7 vulnerability: *"Insecure Data Transfer and Storage"* (c.f. appendix B).

6.4.1.2 IoT application

Our implemented framework is based on the analysis of RIOT applications. The RIOT developers provide an online accessible demo dashboard [30] which is fed by data collected by nodes. The dashboard presents a real-time visualisation of the data of each node and a webcam view of the boards on which the nodes are running.

The nodes' source code is also available online, hosted by Github [54], used later on to verify the results. The nodes run on different boards and have different purposes: gas, temperature, pressure, humidity measurements and so forth. All these nodes use CoAP to send their data, except for the *node_mqtt_bmx280* node which uses the MQTT protocol instead. Since MQTT doesn't support security by its own, we investigated the security of the data transfer on this node.

The *node_mqtt_bmx280* node starts by initialising the MQTT connection and runs three procedures: the *emcute_thread* responsible to receive incoming messages, the *init_bmx280_mqtt_sender* procedure that initializes the sensors (temperature, pressure and optionally humidity) and launches the *thread_publish* that publishes all sensors data every 5 seconds, and finally the *init_beacon_sender* procedure that initialises and launches the *beacon_thread* which publishes every 30 seconds a check message on a specific topic.

6.4.2 Settings

The application itself was compiled from the source code, ready to be deployed on a SAMR21 Xplained Pro evaluation board [41] developed by Atmel, embedding a ATSAMR21G18A microcontroller with an ARM Cortex-M0+ processor.

We determine that the security policy to be verified in the application is to ensure a secured data transfer through MQTT and a secured file storage. This would be guaranteed by using encryption functions before any data sending or storage. RIOT proposes cryptography primitives to encrypt data. The policy could then be formulated as enforcing the use of any of these encryption functions before publishing any data on an MQTT topic or using file writing functions. Hence, we propose the following PRISM policy expression:

```
1 // Ensure encryption : Insecure Data Transfer and Storage
2 // 1 - Ensure encryption: when sending data by MQTT with cipher_encrypt|
      aes_encrypt|chacha_encrypt_bytes|chacha20poly1305_encrypt
3 "always_cypher_encrypt_before_pub" : A [G MQPB=1 => (CPH_ENC=1 | AES_ENC=1 |
      CHA_ENC=1 | CHA_POLY_ENC=1)];
4 // 2 - Ensure encryption: when storing data in a file (fprintf|fwrite|fputs|
      fputc) with cipher_encrypt|aes_encrypt|chacha_encrypt_bytes|
      chacha20poly1305_encrypt
5 "always_cypher_encrypt_before_store" : A [G (FPRNT=1 | FWRT=1 | FPTS=1 | FPTC
      =1) => (CPH_ENC=1 | AES_ENC=1 | CHA_ENC=1 | CHA_POLY_ENC=1)];
```

The "=>" (implication) operator in PRISM states that when the left side condition is satisfied, the right side should be satisfied as well. Interpreting the formula, the model checker ensures that some of the functions on the right side where found on the path leading to the left side functions. In other words, when an MQTT publish operation or a file writing operation occurs, one of the mentioned encryption functions should have been called earlier.

Step	Time
Modelling	264 seconds
Simplification time	<1 second
Conversion time	<2 seconds
Verification time	58 seconds
Total time	324 seconds

Table 6.2 – Duration details of the *node_mqtt_bmx280* node's analysis

6.4.3 Results

Applied to our example, we obtained the following results:

```
"always_cypher_encrypt_before_store": A [ G (FPRNT=1|FWRT=1|FPTS=1|FPTC=1)=>(
    CPH_ENC=1|AES_ENC=1|CHA_ENC=1|CHA_POLY_ENC=1) ]:
Result
true

"always_cypher_encrypt_before_pub": A [ G MQPB=1=>(CPH_ENC=1|AES_ENC=1|CHA_ENC
    =1|CHA_POLY_ENC=1) ]:
Result
false
```

The result is positive for the *always_cypher_encrypt_before_store* property, meaning either a file operation occured but some of the encryption functions were used before it, or that no file operation occured. In both cases, the security property is satisfied fot the tested application.

In the contrary, the result is negative for the *always_cypher_encrypt_before_pub* property, meaning an MQTT publish operation is not followed by an encryption function, and thus, let the data to be transmitted unsecurely through the network.

6.4.4 Verification

By checking the source code, there is no use of any encryption functions in the application. The application involves no file operations, which explains our first result, and thus no need to use encryption functions. However, we can clearly observe that data is published to an MQTT broker, which consequently leaves data transmission unencrypted.

We were able to reproduce the node thanks to the *native* board provided by RIOT. We could intercept the data traffic sent by the application through *tcpdump* and analyse it with the software *Wireshark*. We could clearly read in plaintext the sent data to the remote broker, which validates again the results of our policy-checker tool.

Chapter 7

Discussion

The objective of the study was to evaluate the security of IoT device applications out of their executable format. The settled hypotheses were to have an executable at disposition which included symbols (like function names).

7.1 Achievements

To achieve this, we developed a framework based on model checking, where some security properties should hold in the application model we generated from the executable. The properties are formally expressed in temporal logic and the designed mathematical model is extracted from the executable by static analysis with ICFG reconstruction. The whole process is fully automated. The implemented framework has been proposed as a single independent policy-checker tool on which several experiments have been conducted.

As introduced at the beginning of this work, the IoT has a huge potential for unknown attacks. However, some vulnerabilities are already identified among the most harmful ones, and the OWASP community released in 2018 the top 10 IoT Vulnerabilities (appendix B.1). The experiments conducted in this work tried to solve some of the top vulnerabilities in the IoT and partly cover 2 of them demonstrated in the given examples: *"Use of Insecure or Outdated Components"* and *"Insecure Data Transfer and Storage"*. The work involved real IoT device applications running on the RIOT-OS.

In the context of the SUIT motivating example introduced at the beginning of the discussion, the framework can be placed on the firmware server and check all applications published by the IoT developers against desired security policies. Only applications compliant with the policies could be presented to the IoT devices for updates. In that way, SUIT and our tool addresses also the OWASP top 4 vulnerability *"Lack of Secure Update Mechanism"*.

The implemented framework allows a processing average of around 31 seconds per application checked, not including the graphs rendering which were used for checking the soundness of the model and thus the trustworthiness of the results. The checked applications came from the RIOT example applications repository, which inspires the development of other applications, so this time is expected to grow for any real-world application.

Furthermore, RIOT applications integrate the concept of threads, like in UNIX systems. They are created and then launched by the scheduler depending on their priority and their place in the execution queue. If the execution depends of external factors, like a user input, then it is not possible to know in advance the behavior of the system, limiting the use of static analysis. Our tool considers today any possible combination in the thread sequence, pointing out the worst case scenario, and inevitably over approximate.

All the results were verified manually after the experiment to check the correctness of the methodology and thus the trustworthiness of the results. When possible, a graph visualization of the model was also computed to check that the model was sound and complete.

However, the time spent for the overall check could be improved by spotting the bottlenecks of the processing, with improvements proposed in the next section.

7.2 Limitations and future improvements

Our policy-checker tool is specifically targeting RIOT applications statically analysed from their binary form.

Hence, it suffers from limitations in the tools used for the static analysis of binary code, despite the use of the angr framework, which is currently one of the most advanced available static analysis tools. Indeed, static analysis is a difficult problem still requesting lots of efforts from the research community [70]. Also, static analysis deals with function arguments and especially with function pointers with difficulty, meaning it is not easy to recover the link between a function and its arguments being a function. Calls based on a function pointer may not be understood as such by the static analyser. This implies our tool will consider all functions detached from a direct link to the application as alternative computing. This could provoke false positives as numerous as detached functions.

Furthermore, checking the policies by looking for relevant APIs recovered from the binary is only possible with unstripped executables, i.e. binaries containing symbols. However, stripping binaries is a common technique used in firmware deployment in order to obfuscate the code and/or to reduce the memory space. Indeed, by removing the symbols, the firmware's size is reduced drastically. Nevertheless, to the best of our knowledge, no tool is available to identify common functions or reconstruct functions without any available symbols, that could otherwise be used in conjunction with our model extractor. Therefore, to overcome this limitation, we propose to add the possibility to strip the firmware only after the policy check step. The SUIT process thought for RIOT applications is still under active discussion and could maybe include this specificity.

In addition to that, the policies are formally expressed in temporal logic, hence don't cover all imaginable use cases for an IoT application. For example, if one security property is to detect that a file has been closed after being opened, the variable monitoring this property will still be set even if the file is later reopened, possibly misleading the analysis. A supplementary processing stage during the policy checking step could answer this limitation.

Further improvements could also include the pre-modeling of common libraries expected to be used in RIOT applications and directly use references to these models during the model extraction. This would speed up the modeling phase.

Also, there was a limit on the time spent by the graph renderer when retrieving the program's ICFG, which is independent from the model checking stage and is thus not a proper limitation. It is used to check the soundness of the model manually. The graph rendering is computed using an external tool, so maybe by building our own rendering tool, or using another method to check the model's soundness, could tackle this challenge.

Moreover, some procedures in the final model are not part of the application but the OS. This is due to the recursive search of the callees. While it doesn't impact the model checking process, the extracted model includes additional procedures that aren't necessary. Thus, it could be possible to prune these branches by analysing the OS with an empty application and detect the specific RIOT procedures not to include in the model.

In addition to that, RIOT-OS also works with application written in the C++ language. Additional efforts could be to adapt our tool to recognise C++ functions.

Finally, the IoT still suffer from vulnerabilities, and while our tool cover some of them as explained earlier in 7.1, further work should strengthen the coverage by developing new policies.

Chapter 8

Conclusion

With the expected number of 25 billion devices connected to the Internet of Things in 2021, the connected world is at the dawn of a fully surrounding intelligent environment. At the time of writing, which means 2021 is in 2 years, the population is expected to grow and reach 8 billion people populating the planet in total [64]. In other words, the number of devices will exceed by three the number of people in average, but with the unequal distribution of the smart environments, will be much more concentrated around some parts of the population. These devices will serve to improve the quality of working and personal lives through multiple application domains like the Industry 4.0, smart health or transport. We will be interacting, triggered, and completely plunged in this connected world of devices, whether with tiny RFID tags, robots, pacemakers, or our smart house.

The IoT vision of world-wide interconnected *things* is starting to be perceived. New discovered doors open everyday on new opportunities to fulfill the promise of a smart environment. This revolution passes through technological convergence which still shapes the IoT definition in a daily basis. Achieving trust in such a complex environment is still a hard problem to be solved with global efforts monopolised to catalyse the development of the technology.

Because of the pervasiveness of these devices in our daily interactions, the research community and the industry are both very concerned about the evolution of the cyber risks in the IoT. Recent attacks just showed a small fraction of the potentially devastating impact a coordinated attack could lead on connected infrastructures. They could immediately cause business disasters, or more dramatically, be life-threatening because of attacks on human connected devices like pacemakers or heart pumps. The software patches of field devices are considered essential to maintain a minimum security level in the ecosystem. This work falls within this context, where bringing security is now a paramount.

Frameworks and methodologies for risk assessment don't exist yet to fully assess the cyber risks lying in an IoT ecosystem. Even though huge efforts are driven by international standardization entities together with academic research, the pace of evolution, the constant redefinition of the IoT, and new enabling technologies are some of the additional difficulties of adapting existing frameworks. Combination of several frameworks could cover some use cases, but none of the methodologies are universal enough to embrace them all. Furthermore, periodic risk assessments as conducted nowadays are not suitable for the highly reactive and fast changing IoT environment.

The policy-checker tool developed during this work addresses these needs by providing automatic security policy verifications of IoT device applications. It successfully showed the transfer of a methodology used in the mobile security by adapting the workflow to a new framework suitable for our specific use case. As such, this tool analyses the binary code of an application running on top of the RIOT-OS, it extracts then its security model, and, through the use of a model checker, verifies that defined security properties are met. It could be the presence of specific functions or a certain behavior that are not allowed by a defined security policy. The

tool has been confronted to a real-world scenario using the development process of a RIOT application, first by checking the x86 executable used for development purposes, then checking the final ARM Cortex-M executable aimed to be deployed and used in the real world. The reliability of the tool had been previously successfully checked on a dummy example and 21 real RIOT examples compiled for the x86 architecture.

Indeed, we started the work by introducing the motivating example of a computer assisted security audit process. This would integrate an engine that would monitor each component engaged in an ecosystem. By knowing the twin security model of the deployed application, the engine could continuously check the model against security policies that are developed for known and new discovered attacks. Thus, the system would know the vulnerability status of all deployed applications at any time, which is a first step for setting up appropriate countermeasures in a continuous risk assessment process.

In addition to that, our tool can be used as a supporting tool for the development of IoT device applications, for instance in the frame of the motivating example of the SUIT process, for software updates during the lifetime of the device. The tool could analyse each new firmware present in a firmware server store and check that all applications are compliant with a desired security policy.

The choice of the RIOT-OS to support the applications the tool can analyse is driven by the establishment of a trusted ecosystem. RIOT-OS is open source, in the top IoT OSes used nowadays, is developed by an active community, and is soon engaged in formally proving its kernel. Together with our policy-checker tool, both participate to the creation of a trusted unit, placed in the ecosystem respecting the end-to-end security and security-by-design concepts.

Analysing the executable instead of the source code has many advantages. For example, it answers the dynamic of the IoT application development, where third-party application providers will massively produce applications which source code won't be disclosed. Furthermore, once an IoT device is deployed, it could be running for years. After such a long time, access to the source code may not be possible anymore, but the tool will still be able to analyse it.

However, many challenges remain active around this tool. Among them, there are limitations imposed by the support tools on which our tool is based, like the choice of the static analysis framework that has its own restrictions. Also, the tool is not suitable for other properties than one-time properties, for example applying several times the same APIs (like closing a file) would not be detected properly except from the first use.

Future works could be to apply the workflow on other types of executables, that could address the huge architectural diversity of the IoT devices. It would require to identify the APIs used in the new architectures and adapt the formulas used in the policies.

As a conclusion, the focus of this work was on IoT device application. This doesn't ensure by itself the security of the whole ecosystem. Nevertheless, this tool could thus be used by security risk assessment frameworks relying on the tool's results to compute a system-level risk evaluation. This holistic perspective on the ecosystem is needed to properly conduct the assessments, that will determine the countermeasures that need to be applied and anticipated.

Reaching the step of a completely automated risk assessment process seems not, in our opinion, a realistic goal for the time being. Indeed, still today, many tasks that encounter difficulties to be automated are simpler done by a human. Integrating human intuition and expert experience in the process would be interesting research paths to be explored. However, in the meantime, we believe our work will help securing the IoT ecosystems in order for the IoT to express its full potential and build a better future for all of us.

Bibliography

[1] Carolina Adaros Boye, Paul Kearney, and Mark Josephs. Cyber-Risks in the Industrial Internet of Things (IIoT): Towards a Method for Continuous Assessment. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018.

[2] Andrew Shoemaker. Website of : Imperva, how to identify a mirai-style ddos attack. https://www.imperva.com/blog/how-to-identify-a-mirai-style-ddos-attack/, 2019. [Online; accessed August 7, 2019].

[3] Apple. Website of : Apple, iphone released 2007. https://www.apple.com/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/, 2019. [Online; accessed August 7, 2019].

[4] Alessandro Armando, Gabriele Costa, Alessio Merlo, and Luca Verderame. Formal modeling and automatic enforcement of Bring Your Own Device policies. *International Journal of Information Security*, 14(2):123–140, 2015.

[5] Hany F. Atlam, Robert J. Walters, Gary B. Wills, and Joshua Daniel. Fuzzy Logic with Expert Judgment to Implement an Adaptive Risk-Based Access Control Model for IoT. *Mobile Networks and Applications*, 2019.

[6] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[7] axt. Website of : bingraphvis. https://github.com/axt/bingraphvis, 2019. [Online; accessed August 7, 2019].

[8] Emmanuel Baccelli, Cenk Gundogan, Oliver Hahm, Peter Kietzmann, Martine S. Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wahlisch. RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT. *IEEE Internet of Things Journal*, 5(6):4428–4440, 2018.

[9] Jacob Bergvall and Jacob Bergvall. Institutionen för datavetenskap Risk analysis review av Risk analysis review av. *Institutionen för datavetenskap*, 2015.

[10] Chris Casinghino, J. T. Paasch, Cody Roux, John Altidor, Michael Dixon, and Dustin Jamner. Using binary analysis frameworks: The case for bap and angr. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11460 LNCS:123–129, 2019.

[11] Z. Berkay Celik, Patrick McDaniel, and Gang Tan. Soteria: Automated IoT Safety and Security Analysis. *Proceedings of the 2018 USENIX Annual Technical Conference*, 2018.

[12] Z. Berkay Celik, Gang Tan, and Patrick McDaniel. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT. *Network and Distributed Systems Security (NDSS) Symposium 2019*, (February), 2019.

[13] CINI. Website of : Cini. https://www.consorzio-cini.it/index.php/it/, 2019. [Online; accessed August 7, 2019].

[14] Edmund M Clarke and William Klieber. Model Checking and the State Explosion Problem. Technical report, 2011.

[15] CMU, Cylab. Website of : Binary analysis platform. https://github.com/BinaryAnalysisPlatform/bap, 2019. [Online; accessed September 11, 2019].

[16] CSecLab. Website of : Computer security lab genova. https://csec.it, 2019. [Online; accessed August 7, 2019].

[17] DIBRIS. Website of : Dibris. https://www.dibris.unige.it/en/, 2019. [Online; accessed August 7, 2019].

[18] Eclipse Foundation. IoT Developer Survey Results. Technical Report April, Eclipse, 2018.

[19] ENISA. Enisa gloassary. https://www.enisa.europa.eu/topics/threat-risk-management/risk-management/current-risk/risk-management-inventory/glossary, 2019. [Online; accessed August 7, 2019].

[20] ENISA EU agentur for cyber security. IoT Security Standards Gap Analysis Mapping of existing standards against requirements on security and privacy in the area of IoT. Technical Report December, ENISA, 2018.

[21] E. Baccelli et al. Securing iot operating system software, 2019.

[22] ETSI. Internet of things in 2020, roadmap for the future. https://docbox.etsi.org/ERM/open/CERP%2020080609-10/Internet-of-Things_in_2020_EC-EPoSS_Workshop_Report_2008_v1-1.pdf, 27 May 2008. [Online; accessed August 7, 2019].

[23] Forbes. Entrepreneurs are 'calculated' risk takers – the word that can be the difference between failure and success. https://www.forbes.com/sites/actiontrumpseverything/2013/11/06/entrepreneurs-are-not-risk-takers-they-are-calculated-risk-takers-that-one-additional-wo#7e21f8383e14, 2013. [Online; accessed August 7, 2019].

[24] Gartner. Top strategic iot trends and technologies through 2023. https://www.gartner.com/en/documents/3890506-top-strategic-iot-trends-and-technologies-through-2023, 21 September 2018. [Online; accessed August 7, 2019].

[25] Google. Website of : Google trends. https://trends.google.com/trends/explore?date=all&q=internet%20of%20things,what%20is%20iot,iot%20internet%20of%20things,iot, 2019. [Online; accessed August 7, 2019].

[26] Michael Huth, Peter Burnap, Razvan Nicolescu, David Charles De Roure, Stacy Cannady, Petar Radanliev, and Rafael Mantilla Montalvo. Future developments in cyber risk assessment for the internet of things. *Computers in Industry*, 102:14–22, 2018.

[27] IETF. Website of : A firmware update architecture for internet of things devices draft-ietf-suit-architecture-06. https://tools.ietf.org/pdf/draft\protect\discretionary{\char\hyphenchar\font}{}{}ietf\protect\discretionary{\char\hyphenchar\font}{}{}suit\protect\discretionary{\char\hyphenchar\font}{}{}architecture\protect\discretionary{\char\hyphenchar\font}{}{}06.pdf, 2019. [Online; accessed September 11, 2019].

[28] IETF. Website of : Firmware updates for internet of things devices - an information model for manifests draft-ietf-suit-information-model-03. https://tools.ietf.org/pdf/draft\protect\discretionary{\char\hyphenchar\font}{}{}ietf\protect\discretionary{\char\hyphenchar\font}{}{}suit\protect\discretionary{\char\hyphenchar\font}{}{}information\protect\discretionary{\char\hyphenchar\font}{}{}model\protect\discretionary{\char\hyphenchar\font}{}{}03.pdf, 2019. [Online; accessed September 11, 2019].

[29] IETF. Website of : Ietf suit draft architecture. https://tools.ietf.org/html/draft\protect\discretionary{\char\hyphenchar\font}{}{}ietf\protect\discretionary{\char\hyphenchar\font}{}{}suit\protect\discretionary{\char\hyphenchar\font}{}{}architecture, 2019. [Online; accessed August 7, 2019].

[30] INRIA. Website of : Riot demo dashboard. http://riot\protect\discretionary{\char\hyphenchar\font}{}{}demo.inria.fr/, 2019. [Online; accessed September 11, 2019].

[31] ITPRO. Website of : Itpro, what is the internet of things (iot)? http://cdn2.itpro.co.uk/sites/itpro/files/2017/09/internet_of_things_iot.jpg, 2019. [Online; accessed August 7, 2019].

[32] Johannes Kinder and Helmut Veith. Jakstab: A static analysis platform for binaries - Tool paper. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5123 LNCS:423–427, 2008.

[33] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[34] Krebs on Security. Ddos on dyn impacts twitter, spotify, reddit. https://krebsonsecurity.com/2016/10/ddos-on-dyn-impacts-twitter-spotify-reddit/, 2016. [Online; accessed August 7, 2019].

[35] Christopher Kruegel, William Robertson, Fredrik Valeur, and Giovanni Vigna. Static disassembly of obfuscated binaries. *USENIX Security Symposium*, page 18, 2004.

[36] Sathish Alampalayam Kumar, Tyler Vealey, and Harshit Srivastava. Security in internet of things: Challenges, solutions and future directions. *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2016-March:5772–5781, 2016.

[37] Trishank Karthik Kuppusamy, Lois Anne DeLong, and Justin Cappos. Uptane: Security and Customizability of Software Updates for Vehicles. *IEEE Vehicular Technology Magazine*, 13(1):66–73, 2018.

[38] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[39] Yushi Lan, Fei Jiang, and Hui Yu. Study on application modes of military Internet of Things (MIOT). *CSAE 2012 - Proceedings, 2012 IEEE International Conference on Computer Science and Automation Engineering*, 3:630–634, 2012.

[40] Joseph Galen Lindley, Paul Coulton, Haider Akmal, Duncan Hay, Max Van Kleek, Sara Cannizzaro, and Reuben Binns. *The Little Book of Philosophy for the Internet of Things*. ImaginationLancaster, 2019.

[41] Microchip. Website of : Atsamr21-xpro evaluation kit. https://www.microchip.com/DevelopmentTools/ProductDetails/ATSAMR21-XPRO, 2019. [Online; accessed September 11, 2019].

[42] Emilio Tissato Nakamura and Sergio Luis Ribeiro. A privacy, security, safety, resilience and reliability focused risk assessment methodology for IIoT systems steps to build and use secure IIoT systems. *2018 Global Internet of Things Summit, GIoTS 2018*, 2018.

[43] Dang Tu Nguyen, Chengyu Song, Zhiyun Qian, Srikanth V. Krishnamurthy, Edward J.M. Colbert, and Patrick McDaniel. IotSan: Fortifying the safety of IoT systems. *CoNEXT 2018 - Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, pages 191–203, 2018.

[44] Razvan Nicolescu, Michael Huth, Petar Radanliev, and David De Roure. Mapping the values of IoT. *Journal of Information Technology*, 33(4):345–360, 2018.

[45] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. CHAINIAC: Proactive softwareupdate transparency via collectively signed skipchains and verified builds. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1271–1287, Vancouver, BC, August 2017. USENIX Association.

[46] NIST. National institute of standards and technology special publication (sp) 800-37. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-37r2.pdf, December 2018. [Online; accessed August 7, 2019].

[47] Jason R. C. Nurse, Petar Radanliev, Sadie Creese, and David De Roure. If you can't understand it, you can't properly assess it! The reality of assessing security risks in Internet of Things systems. *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, pages 1–9, 2018.

[48] Jason R.C. Nurse, Sadie Creese, and David De Roure. Security Risk Assessment in Internet of Things Systems. *IT Professional*, 19(5):20–26, 2017.

[49] OWASP. Website of : Owasp. https://www.owasp.org/index.php/Main_Page, 2019. [Online; accessed August 7, 2019].

[50] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. *Proceedings of the 5th Colloquium on International Symposium on Programming*, page 337–351, 1982.

[51] P. Radanliev, D. De Roure, S. Cannady, R.M. Montalvo, R. Nicolescu, and M. Huth. Economic impact of IoT cyber risk - analysing past and present to predict the future developments in IoT risk analysis and IoT cyber insurance. *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, pages 3 (9 pp.)–3 (9 pp.), 2018.

[52] Petar Radanliev, Dave De Roure, Jason R C Nurse, Razvan Nicolescu, and Michael Huth. Cyber Security Framework for the Internet-of-Things in Industry 4 . 0 Cyber Security Framework for the Internet-of-Things in Industry 4 . 0. *Preprints*, (March):0–7, 2019.

[53] Petar Radanliev, David Charles De Roure, Jason R C Nurse, Razvan Nicolescu, Stacy Cannady, and Rafael Mantilla Montalvo. New developments in Cyber Physical Systems , the Internet of Things and the Digital Economy – discussion on future developments in the Industrial Internet of Things and Industry 4 . 0. *Preprints*, (March), 2019.

[54] RIOT developers. Website of : Riot demo applications github repository. https://github.com/future\protect\discretionary{\char\hyphenchar\font}{}{}proof\protect\discretionary{\char\hyphenchar\font}{}{}iot/riot\protect\discretionary{\char\hyphenchar\font}{}{}firmwares/tree/master/apps, 2019. [Online; accessed September 11, 2019].

[55] RIOT-OS. Website of : Riot-os, examples folder. https://github.com/RIOT-OS/RIOT/tree/master/examples, 2019. [Online; accessed August 7, 2019].

[56] Pallavi Sethi Sarangi and Smruti R. Internet of Things: Architectures, Protocols, and Applications. *Journal of Electrical and Computer Engineering*, Volume 201, 2017.

[57] Kewei Sha, Wei Wei, T. Andrew Yang, Zhiwei Wang, and Weisong Shi. On security challenges and open issues in Internet of Things. *Future Generation Computer Systems*, 83:326–337, 2018.

[58] Alireza Shameli-Sendi, Rouzbeh Aghababaei-Barzegar, and Mohamed Cheriet. Taxonomy of information security risk assessment (ISRA). *Computers and Security*, 57:14–30, 2016.

[59] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. In *IEEE Symposium on Security and Privacy*, 2016.

[60] Sogeti Labs. Coming to grips with it in 2016. https://labs.sogeti.com/coming-to-grips-with-it-in-2016-so-stay-tuned/, 2015. [Online; accessed August 7, 2019].

[61] Sowmya Nagasimha Swamy. Applications. *International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)*, pages 477–480, 2017.

[62] UC Santa Barbara, SEFCOM. Website of : angr framework. https://github.com/angr/angr, 2019. [Online; accessed September 11, 2019].

[63] UniGe. Website of : Università delgi studi di genova. https://unige.it, 2019. [Online; accessed August 7, 2019].

[64] United Nations. Website of : Total world population prospects 2019. https://population.un.org/wpp/Graphs/Probabilistic/POP/TOT/900, 2019. [Online; accessed August 7, 2019].

[65] Wikimedia Toolforge. Website of :pageviews of the wikipedia page internet of things. https://tools.wmflabs.org/pageviews/?project=en.wikipedia.org&platform=all-access&agent=user&start=2015-07&end=2019-07&pages=Internet_of_things, 2019. [Online; accessed August 7, 2019].

[66] Wikipedia contributors. Website of : Wikipedia internet of things. https://en.wikipedia.org/wiki/Internet_of_things, 2019. [Online; accessed August 7, 2019].

[67] Liang Xu, Fangqi Sun, and Zhendong Su. Constructing Precise Control Flow Graphs from Binaries. pages 1–22.

[68] Tasneem Yousuf, Rwan Mahmoud, Fadi Aloul, and Imran Zualkernan. Internet of Things (IoT) Security: Current Status, Challenges and Countermeasures. *International Journal for Information Security Research*, 5(4):608–616, 2016.

[69] Koen Zandberg, Kaspar Schleiser, Francisco Acosta, Hannes Tschofenig, and Emmanuel Baccelli. Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check. *IEEE Access*, 7:71907–71920, 2019.

[70] Bin Zeng. Static Analysis on Binary Code. *Tech-report*, pages 1–19, 2012.

Appendices

Appendix A

General risk assessments' methodologies

Their origins range from standard-setting bodies (such as NIST and ISO/IEC) to governments (CRAMM from the UK and EBIOS from France). These approaches are all periodically applied in organizations to assess risk.

- OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation): qualitative, directed towards the most critical assets first following the 80-20 principle (80% of the effects comes from 20% of the causes), tailored for an organization's unique security risk environment, supported online for free by surveys and guidance sheets;
- TARA (Threat Agent Risk Assessment): hybrid, identifies the threats that pose the greatest risk, other risks are ignored, and doesn't include the risk's impact;
- CVSS (Common Vulnerability Scoring System): hybrid, assigns severity scores to the risk associated with vulnerabilities based on a formula depending on several metrics;
- Exostar system (Exostar): qualitative, the company provides services for professionals in order to self-assess the compliancy with NIST SP-800-171. However, the focus is on supply-chain partners, not a stand-alone company, by answering a questionnaire. If the answers to the questionnaire are incomplete or incorrect, its use is likely to lead to incorrect results;
- CMMI (Capability Maturity Model Integrated): risk management is situated at the third maturity level and included in an overall development lifecycle;
- FAIR (Factor Analysis of Information Risk): hybrid, specifically targeting cyber risks, relies extensively on tables. It comes as a commercial software (cost);
- RiskLense : hybrid, built on FAIR as a software solution which constitutes a black box;
- ISO/IEC 27001 (International Standardization Organization): hybrid, combine efforts for standardization of risk assessment, which takes time and makes it difficult to follow the pace of the evolution of cyber risks themselves, voluntary;
- NIST SP800-30 (National Institute of Standards and Technology): qualitative, no tools;
- CRAMM (CCTA Risk Analysis and Management Method) : qualitative, provided as a tool and supported by surveys and guidelines, used notably by NATO;
- EBIOS (Expression des Besoins et Identification des Objectifs de Sécurité - Expression of Needs and Identification of Security Objectives) : qualitative, doesn't provide recommendations or immediate security solutions;

- IRAM2 (Information Risk Assessment Methodology 2): qualitative, focuses on the most significant risks;
- ISRAM (Information Security Risk Analysis Method): quantitative, two independent surveys.

Appendix B

OWASP Top 10 IoT Vulnerabilities 2018

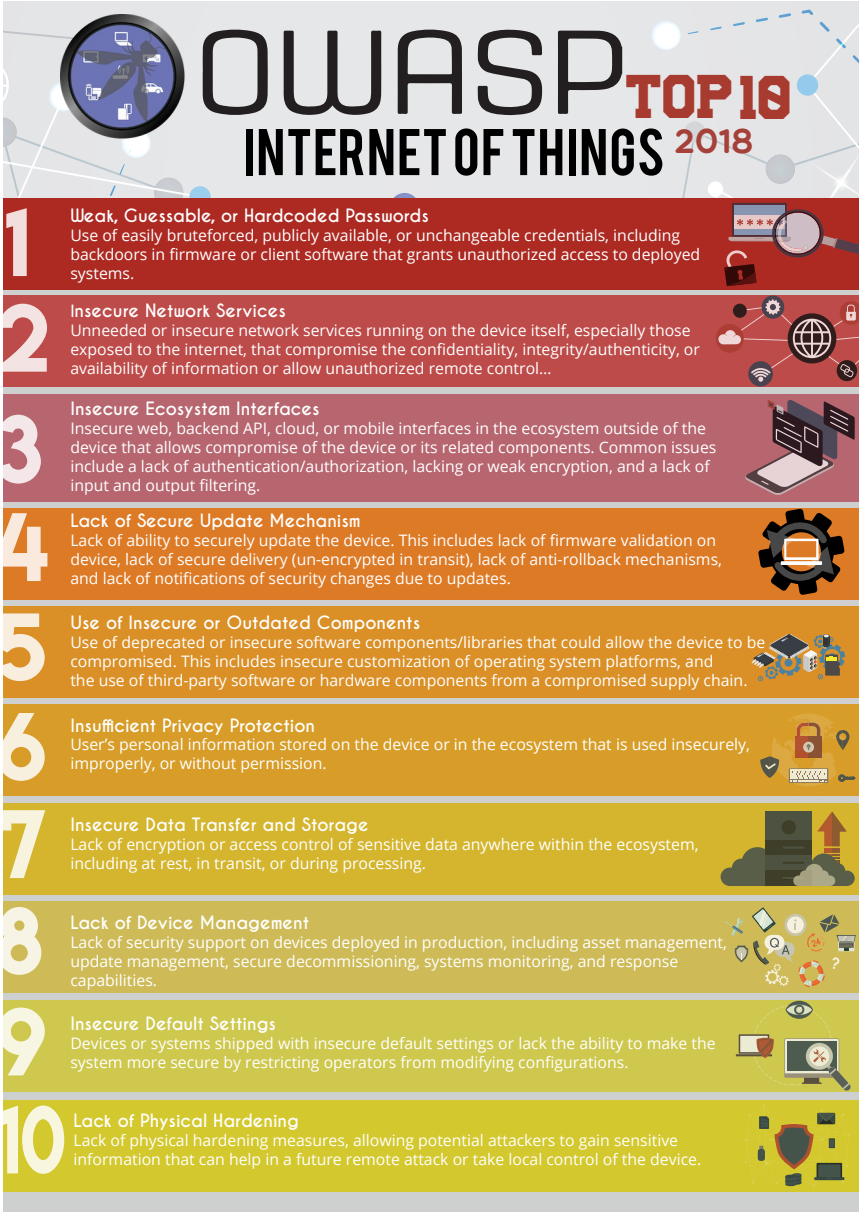


Figure B.1 – The OWASP Top 10 IoT Vulnerabilities [49]

Appendix C

Example files

```
% MCRL2 MODEL OF ELF FILE: thesis_example.elf
act skip;
%<printf>
PRNT;
%<fopen>
FOPN;
%<fclose>
FCLS;
%<fprintf>
FPRNT;
%<fwrite>
FWRT;
%<fread>
FRD;
%<fgets>
FGTS;
%<fgetc>
FGTC;
%<fputs>
FPTS;
%<fputc>
FPTC;
%<puts>
PTS;
%<putc>
PTC;
%<emcute_con>
MQCN;
%<emcute_discon>
MQDCN;
%<emcute_pub>
MQPB;
%<emcute_sub>
MQSB;
%<strcpy>
SCPY;
%<strncpy>
SNCPY;
%<strcat>
SCT;
%<strncat>
SNCT;
%<sprintf>
SPRNT;
%<vsprintf>
VSPRNT;
%<gets>
```

```
GTS;
%<makepath>
MKPTH;
%<_splitpath>
SPTH;
%<scanf>
SCF;
%<sscanf>
SSCF;
%<snscanf>
SNSCF;
%<strlen>
STLN;
%<atoi>
ATI;
%<atof>
ATF;
%<atol>
ATL;
proc
EXTAPI=skip;
%<CFGNode __x86.get_pc_thunk.bx [4]>
H4405035592032427656 = (skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+
    skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+
    skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+skip+
    skip+skip+skip+skip+skip+skip+skip+skip);
%<CFGNode __x86.get_pc_thunk.ax [4]>
H4399155066186473056 = skip;
%<CFGNode do_something [12]>
H4397726874126121931 = H4399155066186473056.PRNT.skip;
%<CFGNode main+0xd4 [10]>
H4399169596965014106 = skip;
%<CFGNode main+0xbc [11]>
H4399195794892733106 = FCLS.H4397726874126121931.H4399169596965014106;
%<CFGNode main+0xb6 [6]>
H4399189273326958056 = (H4399195794892733106+H4397680298621147506);
%<CFGNode main+0xb2 [4]>
H4399210018840388156 = H4399189273326958056;
%<CFGNode main+0x9a [21]>
H4397642149099981556 = FPRNT.H4399210018840388156;
%<CFGNode main+0x86 [15]>
H4397665484065172856 = PTS.H4399210018840388156;
%<CFGNode main+0x7c [10]>
H4397680298621147506 = (H4397642149099981556+H4397665484065172856);
%<CFGNode main+0x6a [18]>
H4399011950964517956 = H4399189273326958056;
%<CFGNode main+0x51 [15]>
H4399036995417465931 = PTS.H4399169596965014106;

init
hide({skip},H4405035592032427656.PRNT.FOPN.(H4399011950964517956+
    H4399036995417465931));
```

example/mcrl2model_thesis_example.mcrl2