



Security Analysis of IoT Systems using Attack Trees

Delphine Beaulaton

► To cite this version:

Delphine Beaulaton. Security Analysis of IoT Systems using Attack Trees. Ubiquitous Computing. Université de Bretagne Sud, 2019. English. NNT : 2019LORIS548 . tel-02893847

HAL Id: tel-02893847

<https://theses.hal.science/tel-02893847>

Submitted on 8 Jul 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE DE DOCTORAT DE

L'UNIVERSITE DE VANNES
UNIVERSITE BRETAGNE SUD

Ecole Doctorale N°601
*Mathématique et Sciences et Technologies
de l'Information et de la Communication*
Spécialité : (voir liste des spécialités)

Par

« **Delphine BEAULATON** »

« **Security Analysis of IoT Systems using Attack Trees** »

« »

Thèse présentée et soutenue à VANNES , le 09/12/2019

Unité de recherche : IRISA Vannes – Équipe Archware

Thèse N° : 548

Rapporteurs avant soutenance :

Chouki Tibermacine, Université de Montpellier

Christine Urtado, IMT Ales

Composition du jury :

Président :

Examineurs : Khalil Drira LAAS-CNRS

Isabelle Borne, IRISA Vannes

Salah Sadou, IRISA Vannes

Régis Fleurquin, IRISA Vannes

Dir. de thèse : Salah Sadou

Co-dir. de thèse : Régis Fleurquin

Invité(s)

Olivier Zendra, INRIA Rennes



Titre : Analyse de sécurité de systèmes IoT à l'aide d'arbres d'attaques

Mot clés : Systèmes IoT, Sécurité, Arbres d'attaques

Resumé :

L'Internet des Objects (IoT) est un environnement qui évolue rapidement et qui permet à des utilisateurs d'utiliser et contrôler une large variété d'objets connectés entre eux. Ces environnements connectés augmentent la surface d'attaque d'un système puisque, entre autre, les risques sont multipliés par le nombre d'appareils connectés. Ces appareils sont responsables de tâches plus ou moins critiques, et peuvent donc être la cible d'utilisateurs malveillants. Dans ce travail de thèse nous présentons une méthodologie pour évaluer la sécurité de systèmes IoT. Nous proposons une manière de représenter les systèmes IoT, couplée avec des arbres d'attaques afin d'évaluer les

chances de succès d'une attaque sur un système donné. La représentation des système est faite via un langage formel que nous avons développé : SOML (Security Oriented Modeling Language). Ce langage permet de définir le comportement des différents acteurs du système et d'ajouter des probabilités sur leurs actions. L'arbre d'attaque nous offre un moyen simple et formel de représenter de possibles attaques sur le système. L'analyse probabiliste sur les chances de succès des attaques est ensuite effectuée via un outil de Statistical Model Checking : Plasma. Nous utilisons deux algorithmes en parallèle pour effectuer cette analyse : Monte Carlo et importance splitting.

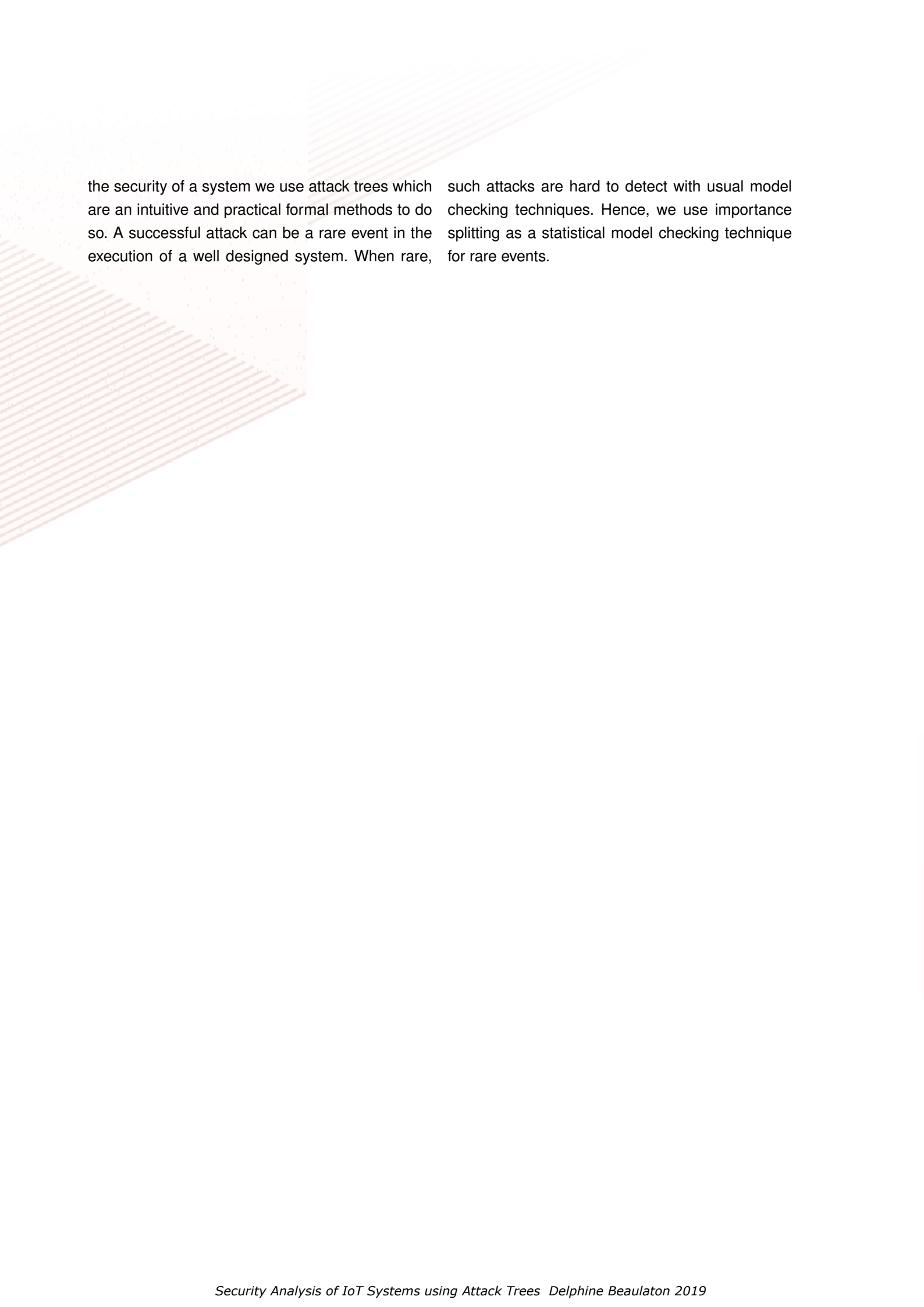
Title : Security Analysis of IoT Systems using Attack Trees

Keywords : Security ; IoT systems ; Statistical Model Checking

Abstract :

IoT is a rapidly emerging paradigm that provides a way to the user to instrument and control a large variety of objects interacting between each other over the Internet. In IoT systems, the security risks are multiplied as they involve heterogeneous devices that are connected to a shared network and that carry out critical tasks, and hence, are targets for malicious users. In this thesis, we propose a security-based framework for modeling IoT sys-

tems where attack trees are defined alongside the model to detect and prevent security risks in the system. The language we implemented (SOML) aims to model the IoT paradigm in a simple way. The IoT systems are composed of entities having some communication capabilities between each other. Two entities can communicate if (i) they are connected through a communication protocol and (ii) they satisfy some constraints imposed by the protocol. In order to identify and analyze attacks on



the security of a system we use attack trees which are an intuitive and practical formal methods to do so. A successful attack can be a rare event in the execution of a well designed system. When rare,

such attacks are hard to detect with usual model checking techniques. Hence, we use importance splitting as a statistical model checking technique for rare events.

Acknowledgements

Tout d'abord j'aimerais remercier mes directeurs de thèse: Salah Sadou et Régis Fleurquin qui m'ont offert cette opportunité de thèse.

Merci à Olivier Zendra qui a été pour moi un chef d'équipe extraordinaire et a rendu énormément de choses possible tout en me poussant hors de mes retranchements.

Je suis infiniment reconnaissante à Ioana Cristescu, sans qui l'algèbre de processus serait resté pour moi un monde obscur.

Pour leur soutien et leur amitié merci à Kévin Bukasa, Armel Esnault, Alexandre Gonzalvez, Tania Richemond, Yoann Marquer, Mathieu Bagot, Nisrine Jaffri, Céline Minh, Rymel Benabidallah, Raounak Benabidallah, Valdemar Neto, Franck Petitdemange, ...

A Pépito, Ramsès et Octana <3

Contents

Acknowledgements	v
1 Introduction	1
2 State of the art	5
2.1 Modelling Techniques for the Internet of Things	6
2.1.1 IoT and Security	6
2.1.2 IoT Language Representations	7
2.2 Attack Representations	8
2.2.1 Tree related representations	9
2.2.2 Attack Graph	14
2.2.3 Others	16
2.3 Statistical Analysis	16
2.3.1 Statistical Model Checking	17
2.3.2 Rare Events Analysis	17
3 Security Oriented Modeling Language	19
3.1 Running Example	20
3.2 IoT Security Oriented Modeling Language	21
3.2.1 Language General Concepts	21
3.2.2 Abstract Syntax	22
3.2.3 Preliminaries for Operational Semantics	24
3.2.4 Operational Semantics	24
3.2.5 Operational Semantics Application on Running Example	25
3.3 Probabilistic IoT Security Oriented Modeling Language	31
3.3.1 Extended Abstract Syntax	31
3.3.2 Operational Semantics	31
3.3.3 Operational Semantics Application on Running Example	34
3.3.4 Concrete Syntaxe	36
3.4 Attack Representation using Attack Trees	38
3.4.1 Attack Tree Graphical Representation	38
3.4.2 Attack Tree Formal Representation	38
4 Simulation Approach	43
4.1 BIP Language	44
4.1.2 BIP Semantics	48
4.1.3 Interaction Between BIP Components	49
4.1.4 BIP Concrete Syntax	52
4.2 IoT Model to BIP	54
4.2.1 Parser Implementation	54
4.2.2 Model transformation	56
4.2.3 Example of Transformation	58
4.3 Formal Transformation	59

4.3.1	Transformation Functions	60
4.3.2	Entity Formal Transformation	63
4.3.3	Bisimulation	67
4.4	Attack Tree to BIP	74
4.4.1	Attack Tree Implementation	74
4.4.2	BIP Monitor	75
4.4.3	Attack Tree Formal Transformation	77
4.4.4	Running Example Monitor	78
5	Experiments	81
5.1	Methodology	82
5.1.1	Statistical Model Checking	82
5.1.2	Overview	83
5.2	Smart Hospital	86
5.2.1	Overview	86
5.2.2	Smart Hospital SOML model	86
5.2.3	Smart Hospital under Attacks	88
5.2.4	Smart Hospital BIP Model	89
5.2.5	Experiments	91
5.3	Amazon Key	93
5.3.1	Overview	93
5.3.2	Amazon Key IoT Model	94
5.3.3	Amazon Key under Attacks	96
5.3.4	Amazon Key BIP Model	97
5.3.5	Experiments	99
5.4	Experiments Conclusion	102
6	Conclusion	103
	Bibliography	105

Chapter 1

Introduction

We often see the Internet of Things (IoT) as a revolution to come. We picture a world in which information is accessible everywhere and human intervention on tedious tasks is made unnecessary.

On your everyday life it means that your house cleans itself and you don't need to worry about the weekly groceries anymore. Based on your consumption habits, whether or not you are expecting guests and taking into consideration your guests' tastes and the occasion, your fridge will order the products. Your errands will then be delivered, regardless of your presence at home, thanks to your smart lock. If you were to go shopping anyway, waiting lines would be old news. You can just go into the store, take whatever you need and just leave. Your account is automatically charged and you are notified, depending on your preferences. Driving to work isn't much of driving as we see it now: your smart car is doing the biggest part of the job. You can prepare for your meetings of the day in the morning or finish your work while checking your kids homeworks on your way back home. Being late because of traffic is not possible anymore since it is regulated by taking into account the main patterns of commuting, extracted from the commuters personal data. No more getting slowed down by the waste truck. Garbage is now automatically routed to the closest sorting centre and processed depending on its nature. Hospitals are never crowded as a vast majority of patients are treated at home, via a real time connection with the health staff. Your doctor has a continuous and real time access to metrics such as your fitness routine and performances, your calory intakes, daily activities, your sleep schedule and quality, etc.

On the industry side, smart farming allows fields to be ploughed and harvested without human physical intervention, reducing costs, accidents and increasing efficiency. Meanwhile, the humidity, air temperature, soil quality and real time forecasts data would be analyzed to create an optimal treatment and watering schedule. Once the cereals have been harvested, the inventories are distributed according to needs of use and ready to be forwarded. The logistic chains being equipped with sensors, trackers and other appropriated devices, the merchandises are safely conveyed, using the most optimal way. Real time monitoring enables users to track them at any time and make sure that the storing conditions remain compatible with the constraints of the product.

We tend to think of this possibility as a vague and distant utopia. But this world is not a far-off future. In fact, it has already started and IoT is getting progressively into our routine. Amazon provides a way for you to receive your deliveries indoors and to allows your guests to get in even in your absence. No need to trust your neighbour with a spare key or to hide it under a rock or a doormat for that: it is made possible with the Amazon Smart Lock. Concerning supermarkets, the Amazon Go allows you, once you have installed the application and registered, to come inside, pick whatever you want and just leave the store without getting by the cashier. Once you have

left, you receive the receipt on your mobile. A dozen of them are already operational in four cities in the US, using as technology a mix of computer vision, sensor fusion and advanced machine learning. The IoT has already entered the automotive market with, for instance the semi project of Tesla. This project is about electric trucks and is meant to reduce energy cost and increase security. It includes an enhanced autopilot and other features such as calling 911 if the driver is not responding. Hospitals are allowing more and more patients to take charge of their health condition at home. They use connected devices and benefit from a real time connection with the hospital staff which can check their metrics at any time.

All these systems provide adapted services with the aim of saving lives, time, money, ... More broadly, they share the willingness of improving the quality and efficiency of user experience. In order to function correctly, the different devices that compose the IoT need to have access to data and to be able to exchange them between each other. The data can be directly provided by a user or a customer or collected through his use of the service. But the data are not only user information, they can also concern the environment. For instance, in the case of smart farming, the data concern the direct surroundings of the crops and are collected by different types of sensors, before being shared with other subsystems.

When talking about the Internet of Things, the innovative and beneficial aspects are often the only highlighted ones. But security issues are real and, if exploited, can cause serious damage. The IoT is composed of a variety of connected devices, and the number of connected objects increase the attack surface of systems. In addition, security experts often try to secure IoT systems the way we used to secure traditional devices. But these new objects have a different behaviour and other constraints. For example, they can be memory constrained, which makes it impossible to install software agents on them. But still, they are connected to the company wireless network and receive and send sensitive data. Hospitals have been the target of numerous attacks, including ransomware, Denial of Service attacks (DDos), etc. In their publication, Mirsky et. al. [51] show how, using penetration tests, they manage to inject and remove lung cancer on computed tomography scans in an active radiology network.

Another issue is that, sometimes, the side effects of a proper use of the system are not taken into consideration. In other terms, new features offered by an IoT system can generate adverse events. We can illustrate this with the use of a fitness tracking application that disclosed the location of secret US army bases. It happened in November 2017 when Strava, a fitness tracking company, released a map of its user activities and exercise patterns in different areas of the world. It showed the itineraries of military personnel on active service, who didn't turn off the tracking option of the application when they went for a run. In this case, the device fulfills its mission perfectly, but the scope of the data manipulated was not considered properly.

All the systems we described previously are built as an interconnection of various kind of devices. These devices exchange data in order to reach a common goal. For instance the different parts of the connected hospital will send each other information in order to treat patients. The critical and confidential aspects of these data can differ and are sometimes not taken into account. For instance, a data that doesn't seem sensitive can allow a malicious user to access other data which are highly confidential. In an attempt to address these security issues we want to be able to calculate the probabilities of success of an attack that is performed on a system by accessing different kinds of information. We then consider attacks where the attacker makes his way towards highly sensitive data by collecting information from different parts of the system that enable him to access his goal.

To do that, we need to be able to model a system and more specifically the information flow among its different subparts. We also need to be able to perform attacks towards this system while carrying out statistical analysis to calculate their chances of success.

The solution we propose can be divided in two parts: modeling IoT systems and possible attacks and the statistical analysis performed on the model.

To model the IoT systems we specified a language: Security Oriented Modeling Language (SOML). SOML is a high level language that allows us to model systems with a certain number of specificities. Details of the model mainly include the communication between the subparts of the system and the data exchanged. The system can also include malicious parties that attack the system. These kind of interactions are described within the model. In order to simulate the behaviour of the system we use a second language: BIP (Behavior Interactions Priority) [26], as well as the BIP compiler. To do so, we implemented a parser that allows us to transform the SOML model into an equivalent and executable BIP model.

In order to monitor the attack happening during the execution, we add a representation of the attack that is external to the SOML model but that becomes an integral part of the BIP model. The attack is specified using an attack tree, that becomes a BIP monitor during the execution.

Now that we have a model that we can run, we add the option to give probabilities to the different actions of the entities. This modifies the conduct of the execution and can increase or decrease the chances of success of the attacker. The inclusion of probabilities enables us to perform statistical analysis on the model. In our case, we use Plasma Lab to carry out Statistical Model Checking on the model. This way, we can compute the chances of success of a given attack towards a specific system, whether they are frequent or considered as a rare event.

This work consists of several research segments that we present in the second chapter. The State of the Art starts by looking into the definitions of the Internet of Things into the literature. Then it explores existing works in terms of security modeling and the different ways of representing an attack. Finally, it delves into the Statistical Analysis currently used, considering rare events as a subpart of it.

The third chapter is dedicated to the presentation of the IoT SOML language. For clarity reasons we start by introducing SOML without the probabilistic aspect, and in the second part of the chapter we add the probabilities. In both parts, using a running example, we describe the abstract and operational aspects of the language. Although the attack representation is external to the model, it is part of the execution. This is why we close the chapter by explaining how we express the attack with the attack tree. The fourth chapter focuses on the transformation from SOML to the BIP language. After introducing the BIP language, we detail formal and technical aspects of the transformation. We also describe the transformation of the attack tree, as well as the monitoring during simulations. The fifth and final chapter is devoted to the experiments. We use two real examples of IoT for them: a smart hospital and the Amazon Smart Lock. The attack trees of both examples contain a compilation of real attacks that have been carried out on the systems either for testing or for malicious purposes.

Chapter 2

State of the art

This work addresses security issues in distributed environments, and more specifically in the Internet of Things. We aim to represent IoT systems in an abstract but realistic way as well as possible attacks towards them. These representations are then used to carry out statistical analysis to calculate the chances of success of the attacks on the systems.

Therefore, this work can be divided into three main aspects: the system model, the attack representation, and the statistical analysis. This chapter reflects these three main leads that compose this thesis.

We start by giving an overview of the Internet of Things in Section 2.1, before addressing the security challenges that it is currently facing. Section 2.2 presents multiple ways of representing attacks. Finally, Section 2.3 presents different ways of performing statistical analysis methods for rare events as well as similar approaches.

2.1 Modelling Techniques for the Internet of Things

Aztori et. al. [3] define the Internet of Things (IoT) as a "pervasive presence around us of a variety of things or objects which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals". They also define different IoT paradigms, depending on the scientific community that consider it. But the issues that arise with the omnipresence of the IoT are the same for everyone and concern, among others, security and privacy. In this section we searched for existing work that addressed these security issues, and in particular through secure representation of the systems. We start by giving an overview of security oriented representation of IoT systems in Subsection 2.1.1. Then in Subsection 2.1.2 we give a global vision of existing languages to represent IoT systems, whether they are security oriented or not.

2.1.1 IoT and Security

In their work, Q. Jing et. al. [34] argue that IoT systems are composed of three successive layers: the perception layer, the transportation layer and the application layer. The perception layer concerns information collection, object perception and object control. Transportation layer is responsible for giving access to the first layer, and data transmission. Finally the Application layer's role is to support business services and realize intelligent computation and resource allocation. This vision of IoT systems allows to detail security features and requirements, and therefore to adapt the security solution, depending on the target level of the device.

All the description aspect of this work is based on the technical description of the system architecture. Indeed, by dividing the overall architecture into layers, it becomes easier to apprehend and address the vulnerabilities, depending on their level. On the other hand, detecting cross-layer vulnerabilities requires to see the system as a whole and, in this case, finding the adapted security measures might be more difficult.

Because of its interconnected nature, IoT systems are vulnerable to a large spectrum of attacks. Indeed, we have to consider that only one unsecured device can affect the security of the overall system. A way of identifying new attack surfaces using a visual representation has been proposed in [59]. In this work, IoT security is not considered as a binary concept but rather as a spectrum of device vulnerability. They then offer a new visual grammar to describe IoT systems at a high abstract level. Their representation is presented as a three-tiered structure constructed as follows: the first layer is the high level representation that uses the visual grammar in order to offer a model of the IoT system. The second layer shows the object profiles and connects the first and the third layer which provides the implementation details of the system. The second layer maps out the high level representation with the low level representation that gives us technical details about the connected objects that constitute the system. To build the high level model, they use the device characterizations of all objects that are part of the system. Then, they mapped the device descriptions to one or several existing devices, and finally, they generate a high level representation. The whole model is meant to provide information about the objects, the structure they form, as well as their activities and relationships.

This work represents a link between the technical characterization of the devices that compose an IoT system and an abstract representation of the system. But the security aspect of this work is based on the possible vulnerabilities of each device, from a technical point of view. The abstract representation is just meant to facilitate the

vision of the overall environment before delving into the features of each device.

2.1.2 IoT Language Representations

In this subsection we separate two types of representations: the one that were created for security purpose and the others.

IoT modeling

A visual representation based on the UML notation exists in [22]. The goal was to implements a language that is powerful enough for the professionnals while being user friendly and easy to read. In this language, the basic element of an IoT system is called a "Thing". The thing can be either real or virtual. When things are arranged in collections, they compose a "subsystem". Things can contain items that can be inputs, outputs or components. All these concepts that compose the language possess a virtual representation inspired by the UML class diagrams. Even though this virtual representation is not security oriented, as the SOML (Security Oriented Modeling Language), we could use it as an inspiration to create a visual representation of our language. Indeed, for now, to describe a system using SOML, one has to know the language syntaxe. By providing an interface with visual elements we could, as a future work, consider to generate the system model automatically from a visual interface.

Another language, GroupeSens-L [1] offers to represent Group Activity Recognition. The goal is to model the physical activity of a group using an EBNF based modeling language. It allows to model the group activity with their conditions and constraints, to make vary the level of details and to generate a billing model.

In [6], the authors use a meta model in order to help developpers build their IoT system as part of a model driven development process. From the meta model, the developer builds its own model and can automatically generate Java code, which is meant to be used as the departure of its IoT system implementation. The meta model is built around "Human Object View". It means that it represents the interconnection between human and physical connected objects as well as the way humans use these objects to expand their communication capabilities. This point of view shares our vision of the communication between human and objects, but as it is not security oriented, it does not make appear verification, of the critical aspect of the exchanged information, which is essential in our work.

Another approach is the IRON language [13], which is an ECA (Event-Condition-Action) based language with a formally defined semantics, meant to identify incorrect behaviours of an IoT system. The IRON language has a static part and a dynamic part. The static part concerns the variable declarations. The variables are the connected devices that are described using an identifier and a name. The static part also defines the constraints that ensure the validity of the exchanged information. The dynamic part is about the ECA rules that descibe the behaviour of the devices. An Event-Condition-Action rule indicates how an action is performed during an event when the condition is filled. The formal semantics is able to ensure the reliability of the results of the execution. Even though this modeling language allows the execution of IoT model to guarantee the correct fonctionning of the system, the errors are only considered from a logical point of view. The attacks towards the system, as well as malicious users or actions are not considered.

An UML extension language called SOAML4IoT [17], can be used to model SOA-based IoT applications. SOAML4IoT is an extension of the SoaML (Service oriented architecture Modeling Language) which is already an UML extension for SOA applications. This extension of SoaML is enriched with fundamental concepts of the IoT models such as platforms, applications and networks. Each of these concepts possess communication, exchange, ... capabilities that are specialized as tasks. As for the previous work, this language is meant to be used for a static modeling during the conception, and cannot be used in our approach but it is interesting to notice that the same concepts (connected device, communications, ..) are highlighted for each IoT-specialized modeling language.

IoT secure representations

The IUT Standard Specification and Description Language (SDL) [63] is an executable language, meant to model and simulate the functioning of IoT systems or other distributed environments. Using SDL, one can model an IoT system as a set of connected agents that communicate with each other. The SDL simulator allows to verify the behaviour of the system before it is deployed. The abstract machine semantics of the language helps define the communication architecture of the system (communication elements with different level of abstraction) and the complete behaviour of the system. But if SDL allows to model the behaviours different actors of the system, whether they are malicious or desired, it does not address potential attacks towards the system. Its only concern is to detect security issues considering the normal run of the system.

This approach and our work have a similar approach First, it uses a high level model of the system and it allows running simulations in order to find anomalies in its behaviour. But our approach is meant to simulate one or several external actors attacking the system and observe their progression. SDL doesn't take into account possible external attacks. The security issues that are considered here are the one that can occur during normal operation of the system. In this case, the model operates as a realistic system replica and allows to correct issues before deploying the environment.

IoTSec [58] is an UML based language supposed to encapsulate security knowledge to model IoT systems. The work uses the model based system engineering approach. It is based on the idea that to resolve safety issues in the IoT, we need to perform security design as well as vulnerability and threat analysis before implementation. This UML/SysML extension redefines several UML diagrams to model security problems in IoT systems such as the class diagram, the sequence diagram, the state machine diagram, ... This work tends to address security issues in the system by modeling them during its conception. The models are not meant to be executed and displays vulnerabilities and threats possibilities. Therefore, these techniques could not be used in our approach.

2.2 Attack Representations

Modeling attacks, threats, faults, etc. anything that can be a risk for the proper functioning of a system has always been a real concern in research. The objectives can either be to prevent it, detect it or as it is our case: estimate the probabilities for an attack to happen. There are a variety of options with different possible applications. Following [42] we separate them into three main categories: Tree related representations, Attack Graph and Others.

2.2.1 Tree related representations

This subsection describes all the attack representations that use a tree-like representation. We first start with the attack trees, which are the way we chose to represent the on going attacks towards the system we audit. Indeed, attack trees have the advantage of being easy to understand thanks to their intuitive representation. It is also possible to modify the amount of information they contain, depending on our needs for the analysis. Several alternatives, based on the attack trees exist such as the defense trees, threat trees that we detail in this subsection. They extend the attack trees with informations such as the defense information or economics indexes for deeper analysis. In this work we didnt need to extend the attack tree as the information necessary for the analysis are contained in the model of the system. But it is possible, as a future work to consider conducting different or further analysis on the IoT systems. In this case, another tree-like representation can be used, that would contain different types of information concerning the attacks towards the model.

Attack Tree

Attack trees we first defined by Bruce Schneier [62] as a formal way to describe the security of a computer system, and to be used in order to make security decisions. One of the benefits were then to be able to make the attack trees evolve easily as the system was developed and the security concerns changed. The aim of an attack tree is to cover more or less exhaustively, depending of the needs, possible attacks towards a system. We can see an example of an attack tree defined by Bruce Schneier in

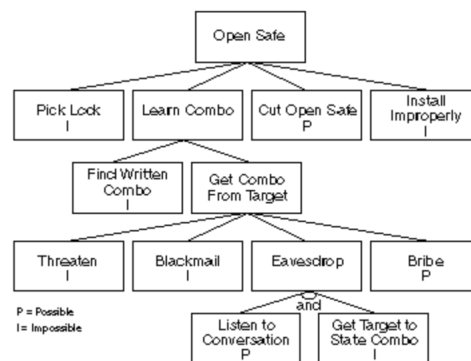


Figure 1: Attack Nodes

FIGURE 2.1: Example of an attack tree as defined by Bruce Schneier

Figure 2.1. The root node represents the goal of the attack: here it is about opening a safe. There are several ways to reach this goal, such as pick the lock or cut the safe open. The annotations on the nodes of the tree can differ, depending on the needs. Following the work of Schneier, Mauw et. al. propose a formal semantics for the attack trees [47]. In their work, they formalize the concepts introduced by Schneier and define writing rules to standardize the creation of attack trees. This can be useful, for instance in computer aided threat analysis.

Due to their simplicity, attack trees can be used in various environments and they allow users to have a global overview of the attack. But because of this, one can argue that attack trees do not have a sufficient level of technical specificities to properly allow to represent cyber threats [23]. But T. Tidwell et. al. [65] found a way to overcome this issue. Indeed, they propose an attack visualization system using attack trees in which they are used to represent Internet attacks described as "multistage

attacks". An attack tree is defined with weighted nodes, indicating the likelihood of their success. To be able to add technical details, the attack tree is used in conjunction with a parametric extension. This means that each tree is associated to a unique specification template that captures information like the pre and post conditions of the attack. This way, they can automatically link an attack to a given vulnerability of a system and this helps them in performing vulnerability assessment and attack prediction. This approach is interesting because it allows to link vulnerabilities of a system and one or several possible attacks and add a low level point of view. But the attack trees are still designed manually by the security expert and therefore are inspired by the existing, they can't be used to discover unknown attacks towards a system.

Defense Tree

Also called Attack-Defense Tree. Defense trees appear for the first time in the work of Bistarelli et. al. [8] as an extension of attack trees. The authors aim to provide security managers a methodology to perform an economic evaluation of the risks via an assessment of the possible countermeasures. The authors propose to extend attack trees by adding countermeasures to each node representing a vulnerability and label the countermeasure with an economic index. This method combines qualitative and quantitative approaches. The qualitative view corresponds to an exhaustive evaluation of the risks using security scenarios and represented as a tree. The quantitative view is meant to give precise measures of the risk and consists in assigning quantitative indexes to each threat described in the tree. By extending attack trees the authors aim to compensate the fact that attack trees do not take into account countermeasures and the cost they represent. This method allows an economic evaluation of the threats as well as their effectiveness. It can help decide if they are profitable and effective. In [9] Bistarelli et.al. go further in the development of the quantitative approach. The economic labels added to the tree are in fact two utility functions: the ROI (Return on Investment) and the ROA (Return on Attack). They then transform an attack into a strategic game with on one side the attacker and on the other the defender. The ROI and ROA allow to calculate the Nash Equilibrium that represents the best strategy possible for both sides of the game. One conclusion reached by the authors is that if installing countermeasures is always worth it for the defender, multiplying the countermeasures does not necessarily represent an increased benefit for the company. Finally, [38] introduces a formal representation of defense trees. The authors defined ADTerms (attack-defense terms) on which the language is based and that allow to formally represent attack-defense scenarios.

Threat Tree

Although they are sometimes assimilated with attack trees, for instance in [52], we consider them to be different. Threat trees express vulnerabilities that can be exploited and lead to a main failure. They can be used in order to express the weaknesses exploitable by a third party to conduct an attack, and therefore express attack scenarios. In fact they don't express the direct actions of an attacker but rather weaknesses that could lead to an exploit. The threat modeling process expressed in [52] and [46] is a four step process. First the system must be decomposed in order to identify its main components. Secondly, these components become the threat targets, it is necessary to identify all the threats. Thirdly the threats must be classified, usually by descending security risk. Finally, users determine how to respond to threats. In their

approach [46], the threat trees are considered to be a variation of fault trees for safety analysis and are used to test whether a software system is secured or not. To this end, they automatically generate security tests as well as valid and invalid test inputs in order to expose vulnerabilities within the software. As for [55] they developed a methodology to capture a large number of cyber threats towards large systems. The aim of their work is to define what the threats are, catalog them and then assess what countermeasures can be set up. The first step is from a risk assessment process, to build a cyber threat tree. This stage is supported by CyTML (Cyber Threat Markup Language), a formal language defined by the authors to represent cyber threat trees. They define their tree as a superset of fault trees and attack trees. Then the tree representation is transformed into a MDD (Multiple-Valued Decision Diagram). This MDD structure provides an easy way to go through the cyber threat catalog and then identify the best way to protect the components of the system.

Fault Tree

Fault tree analysis is one of the most used techniques in the industry to evaluate the risks within a system. Static fault trees were introduced by the Bell Labs in the 1960s for the analysis of a ballistic missile [67]. The traditional fault tree aims to model how combination of components failures can lead to system failures. An application of fault trees in the industry concerns for instance the measure of the risks faced by employees in their workplace environment.

A fault tree, of which we can see an example in Figure 2.2, drawn from [60] is

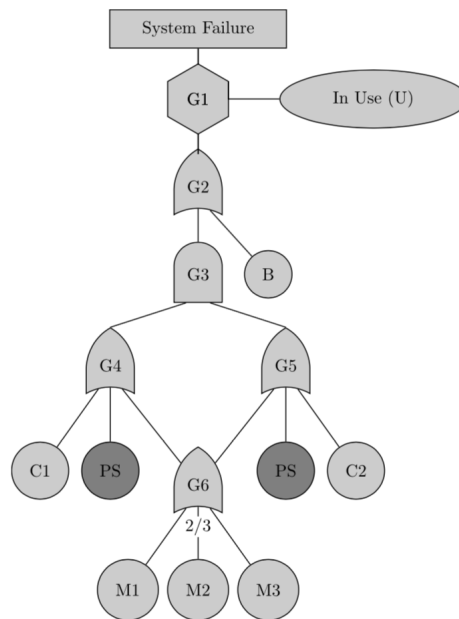


Fig. 1 – Example FT of a computer system with a non-redundant system bus (B), power supply (PS), redundant CPUs (C1 and C2) of which one can fail with causing problems, and redundant memory units (M1, M2, and M3) of which one is allowed to fail; failures are propagated by the gates (G1–G6). PS is somewhat darker to indicate that both leaves correspond to the same event.

FIGURE 2.2: Example of a fault tree

composed of two types of nodes. The leaves of the tree represent the model component failures. These kind of events can be either called "basic event", which means

that they can happen spontaneously, or "intermediate events" and therefore are the consequence of one or several other events. The internal nodes, called "gates" express how the failure propagates throughout the system. The root node of the tree is the "top event": it is the event that is under review. The different nodes are graphically represented with different shapes as we can see in Figure 2.3 from [60], depending on their types.

As for the attack trees, fault tree can be expanded: for example dynamic fault trees add a temporal dimension. This allows to show that even if the same component fail, some failure sequences are more dangerous to the system than others.

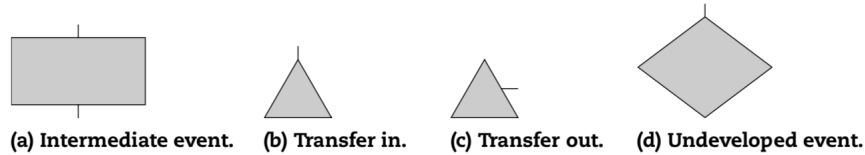


Fig. 2 – Images of non-basic events in fault trees.

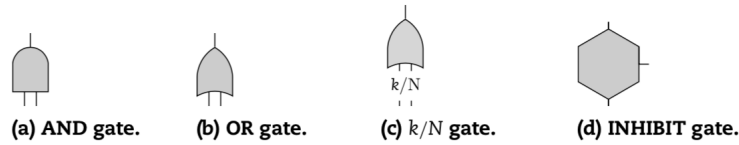


Fig. 3 – Images of the gates types in a standard fault tree.

FIGURE 2.3: Different kinds of nodes in a fault tree

But fault trees can also have security purposes. In [29], fault trees are used as a way to implement intrusion detection systems for software systems. The method is the following: first, an intrusion is decomposed into seven stages that include a reconnaissance and then the exploitation of the vulnerabilities identified in order to extract and modify data. Each of these stages are represented as a distinct fault tree that contains all the combinations of events that have to happen in order for the stage to be complete. This means that, at least, seven fault trees (one for each stage) are necessary to represent an intrusion.

Even though fault trees can have security applications as we saw in [29], fault trees are not suitable for our needs. Indeed, we want to model the actions of an attacker and the only information on the actions that we need are what information are exchanged. The original purpose of fault trees were to model technique failures in industrial components and therefore it focuses on technical events that can happens within a system which is not our preoccupation.

Attack Nets

The idea behind the attack nets [48] is to use Petri Nets [57] (see Subsection 4.1.1 for a detailed definition of Petri Nets) to model attacks for penetration testings. Places of the Petri Nets are composed of security relevant states of the entities of interest, and the transition, built from flaws of the system shows input event, commands or data that allow to access those states. Tokens are used in the Petri Nets to indicate the progression of the attack. Tokens can even be labeled to differentiate different attackers and their actions. The idea behind this approach is by aggregating all the Petri Nets concerning on system, to show the cumulative effects of the flaws, given

that each transition represents the exploitation of a flaw. Compared to attack trees, the Petri Nets are presented as a more expressive and direct way to show the actions of the attacker. In this spirit, a method of collaborative attack modeling: ATiki [64] was implemented. ATiki extends a WikiWeb-like system that acts as a vulnerability database. An ATiki model is then composed of conditions that are the places of the Petri Nets and that describes the system properties. As for the transitions, they are explicated by the wiki pages and open for collaborative improvement. This work is supposed to facilitate cooperation between security experts of various areas and therefore allow the discovery of new vulnerabilities. But Petri Nets can also be used in a more technical approach. In [30] attack nets are used jointly with software fault trees for anomaly detection in software based systems. Petri Nets are automatically generated from software fault trees with an intrusion detection system.

Threat Nets

Based on the desire of improving expressiveness and semantics of the existing threat modeling techniques, Mirembe et. al. [50] decided to use Petri Nets for threat analysis. They consider that to be effective, threat analysis must not be attacker centric or threat centric. Therefore, their method captures both system attributes, by incorporating flaws of the system design and development, and attacker profile with his background knowledge and his time specific actions. In this work, the use of Petri Nets is similar to the attack nets that we previously defined but extended with background information about the system and the attacker. The use of Petri Nets allows to see the threats at a micro level and then avoid propagation errors that are possible with a high level of threat modelling.

Protection Trees

Edge et. al [20] define a version of the protection tree that is built directly from a given attack tree. The overall structure of the protection tree stays the same: it is an AND/OR tree structure. The root node also stays the same as in its corresponding attack tree. But the goal when building a protection tree is to eliminate all attack paths of the attack tree, so the rest of the tree differs. To each possible attack represented by a path in the attack tree, the protection tree presents one or several countermeasures expressed as a node or as a path. Each node of the protection tree is labeled with the cost of the measure. This way, security experts can compare the cost of the different countermeasures. This approach is similar to what we defined in defense trees previously. The difference is that the protection tree is built in reaction to a specific attack and the protection tree only includes defense information. The aim is to help security experts make their decision for allocating their limited defense resources.

Vulnerability Trees

Vidalis and Jones [66] propose to represent vulnerabilities for security assessment using vulnerability trees. The graphical representation of vulnerability trees is similar to that of fault trees but their content vary widely. The top node of the vulnerability tree is the "top vulnerability" or "parent vulnerability". The child nodes of the tree represent the "child vulnerabilities": there is one for each way to exploit the top vulnerability. For each child vulnerability, there are one or several steps that the threat agent has to perform in order to exploit the child vulnerability. Then, once each child is decomposed into steps, each branch of the tree figures a detailed way to exploit

the top vulnerability. The nodes also contains information such as "TTE": Time To Exploit that depends on the number of steps necessary, or a description of the vulnerability. The aim of this work is to help a security expert analyze different attack scenarios and is meant to help him choose adequate countermeasures, by examining vulnerabilities and their relationships.

For analysis purpose we need to be able to divide attacks into stages. We also have a varying amount of information that we need to know for each of them. Attack trees allow us to model the different ways to conduct an attack. There are also easy to modify, depending on what we need to know for each step of the attack. If we considered adding features in our process such as countereasures, costs of the attack and of the defense, ... We could consider using an extended version of the attack tree such as a defense tree as the structure and fonctionning remain the same. To get additionnal information about defense we could also consider using protection trees, as they would be generated from the attack tree we already have.

Threat nets capture some of the system properties. Our work has for characteristics to clearly separate the system attributes and the attack features. Using a threat nets would create an unnecessary overlapping of information. The same goes for the attack nets which aim to show the cumulative effects of the flaws and the threat trees that lists all possible exploits within a system or vulnerability trees. In a future work, we could explore the possibility of generating an attack tree automatically from the referenced exploits, flaws or vulnerabilities of the system, but we would need to consider low level aspects of the system which is not our current goal. Finally, fault trees are widely used in the existing in similar approaches but their applications are very different to the attack trees'. Indeed, it is used for risk analysis in industrial environments, which is very different to attack analysis.

2.2.2 Attack Graph

Concerning the attacks graphs, several approaches are possible.

"General" Attack graph

For instance, [54] uses attack graphs in the context of network security. The idea is to build a tree that, given a set of attack goals, references the necessary security conditions to guarantee the safety of the network system. The building of the tree starts with a sequence of exploits. The conditions are derived from these exploits and they can be of two types. First they can be the exploits pre conditions and are presented as initial conditions existing in the system. They represent the most urgent security failures in the system to correct to ensure safety in the network. The other type of conditions can be the pre and post conditions of the system, during or after an attack. They can be influenced by the exploits performed within the system and therefore the network administrators don't have total control over them. So the graph basically represents the dependencies between security conditions and exploits. Using it, the network administrator has a vision of the measures to be taken in order to protect his system as they go beyond possible attacks. In the ADEPTS methodology [24], attack graphs are used to implement an automatic containment response to an intrusion in distributed systems. The aim is to restrict the effects of an attack and allow the users to keep using the fonctionnalités even though the system is currently under assault. The attack graph, called I-GRAPH, lists several attack goals and the

different steps to achieve them. The different goals can have one or several common steps and therefore the graph shows the inter dependencies between them. From the graph, the system can see where is the attack at and anticipate how it will most likely spread. This allows the system to set up an automatic response to contain the intrusion.

As we can see on these two applications of the data graphs, they can contain different types of information, depending on the need of the security experts.

Alert Correlation Graphs

Correlation trees were firstly described by F. Cuppens et al [18] as a way to reduce the number of alerts in Intrusion Detection Systems (IDS). To do so, they combined several IDSs into a cooperation module. This way, when an intrusion occurs, the system recognizes the different alerts that correspond to a certain kind of intrusion. Then, by regrouping them, it generates a more global and synthetic alert that sums up all of the necessary information. The displayed information contains an overview of the current situation (what was performed by the attacker), the security state of the system and possible options of the attacker next move. In order to anticipate the attacker's behaviour, the module uses a correlation function. This function maps out an existing database of attacks to the alert database. The attack representation includes the pre and post condition of the attack as well as several scenarios such as the attack, detection and verification scenarios. Then from these two separate databases, each generated alert will be correlated to one or several possible attacks. The security administrator will therefore receive global and synthetic alerts that are meant to help him take adapted decision in order to protect the system. A similar approach exists in [53] with the aim of correlating alerts, using prerequisites of intrusion. Their approach is based on the observation that most intrusions are correlated to one attack as they represent different stages of it. The initial stages are meant to facilitate or to allow the following ones to happen. In this work, a graph based formalism allows them to treat a large number of data concerning alert information and attacks.

Privilege Graphs

In their work, M. Dacier et. al. [19] use a directed graph to assess security violation due to privileges accesses. This work is based on two main concepts: protection state and authorization scheme. The protection state refers to the sets of rights held by each individual of the system. The authorization scheme shows how the set of rights evolve, how privileges can be granted within the normal functioning of the system. These information are synthesized in a directed graph called privilege graph. Each node of the graph contains a subject, an object and a set of rights. They represent a set of privileges that can be granted by an individual on a set of object. The edges of the graph indicate the authorization scheme: what privileges can be acquired throughout the graph. An edge between a node N_1 and N_2 shows what additional privileges can be acquired in the node N_2 when the ones of the node N_1 have already been granted. Therefore, a path in the graph represent all the privileges that an individual can acquire in the normal functioning of the system. Given a set of states that violate system safety, the graph enables a security expert to identify the paths of the graph that are in direct conflict with the security constraints. The graph gives a boolean answer to a safety problem by answering the following question: is an unsafe state reachable or not?

In our work, we measure the progress of the attack by the level reached in the attack tree. Therefore having several root nodes as in the attack graph would be a problem to get the chances of success of a specific attack. As for the alert correlation graph, the application domain are very different as it is meant to avoid or reduce intrusion within a system. Finally, the privilege graph's philosophy intersects with ours as we try to show how an attacker grants privileges and critical information throughout its communications with other actors of a system. It would be possible to add probabilities in the privilege graph in order to calculate the probabilities to reach an unsafe state. But in our work, having a separate representation of the system allows us to have a richer vision of the system with information such as the protocols used for communication.

2.2.3 Others

Finally, there are other approaches that couldn't fit in a tree or graph category. In this subsection we are giving a brief overview of influence diagrams [31], kill chains [45] and diamond model [14]. The two following works are about intrusion analysis. The idea behind the kill chains [45] is to completely understand an intrusion in order to set up an adapted line of defense. The kill chains system uses indicators to describe the attack and anticipate the attacker's behaviour that is called a "kill chain". Then, considering the defense capabilities, it helps choosing the most appropriate one and finally to measure its efficiency. This work is about having a very specific and adapted way of describing the attacker's moves and the damages they create in the audited system.

The diamond model [14] is a highly adaptable model meant for intrusion analysis, meant to express insider and external threats. It uses a formal description that can be generic and flexible and, by allowing to capture important information about the intrusion, eases decision making. The diamond model represents the actions of an attacker as events that have characteristics such as a timestamp, a result, resources, etc ... Figure 2.4 shows an extended diamond model. It shows the core features of an intrusion event: adversary, capabilities, infrastructure and victim. It allows to link these features to fully understand the extent of an intrusion.

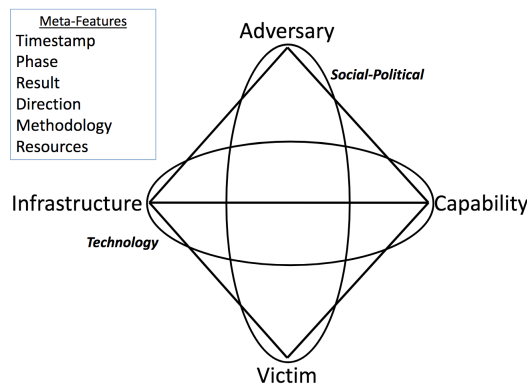


FIGURE 2.4: The diamond model of intrusion

2.3 Statistical Analysis

As the end goal is to calculate the chances of success of an attack, our work includes a statistical analysis part. We performed this analysis using a statistical model checker.

In this section we give an overview of what model checking, how it evolved and other verification techniques in Subsection 2.3.1. We follow by explaining what is a rare event in Subsection 2.3.2 as some of our attacks are rare events.

2.3.1 Statistical Model Checking

Model checking is an automatic verification technique destined to finite state systems. The original model checking method was developed by Clarke and Emerson in 1981 [16]. The goal was to automatically verify finite-state reactive systems. The specifications to check were expressed using temporal logic, and the systems were modeled as state-transition graphs [15]. But these first model checkers were limited to small examples as they experienced state explosion problems on larger models. This difficulty was overcome by the use of ordered binary decision diagrams [11]. With this new technique, the transition relation of the system were expressed using a boolean formula and then converted to a binary decision diagram, that allowed to have a very concise expression of the transition relation. This new method is called symbolic model checking [12].

With these methods is it impossible to quantify the impact of changes made in the model. Indeed, making changes in the design can completely change the outcome of the verification. To be able to quantify the impacts of these changes, a finer analysis is necessary. Using probabilistic model checking, instead of a boolean response (the system verifies the specification or not), it is possible to measure the likelihood of a system to satisfy a property, using tools such as PRISM [39]. It is also now possible to add features such as real time in specifications. In fact, in some cases it is equally important to know when the situation occurs: this is what motivated the creation of the timed model checking methods with tools such as UPAAL [43].

Other verification techniques exist such as for instance theorem proving [35]. Theorem proving is a formal method of verification in which we use specifications, written using a mathematically based notation, and then we verify if the specifications are satisfied. Completely checked systems using this method can be produced when the systems are small. Compared to this method, model checking is completely automatic and faster. Other methods [28] include RSML (Requirement State Machine Language) that is based on hierarchical finite state machine and decompose the specification into a state hierarchy. This decomposition makes it easy to use on large and complex systems but it is not flexible in language design and we couldn't use it for statistical analysis. Another formal method is Independent Verification and Validation, used to analyze partial informal specifications. It can be used for instance on fault detections.

When comparing these verification methods, the result is that model checking offer the biggest flexibility in terms of language specifications. It is also the only method that allowed us to perform statistical analysis.

2.3.2 Rare Events Analysis

A rare event is an event that has a very low probability to happen. Rare events simulation is important in the world of physics to answer question such as "what is the probability of collusion between a satellite and a spatial debris?". Two main algorithms can be used to perform rare event simulations: importance splitting and importance sampling. Both are variant of the Monte Carlo algorithm citeimportance-Splitting. In our work, we use the importance splitting method that we explain in more details in the Subsection 5.1.1. The first mention of importance splitting is in

[36], where the algorithm is used to calculate the probability that neutrons would pass through certain shielding materials.

Concerning the importance sampling, we find an approach similar to ours in [61], but transposed into a different area. Instead of security, this work tends to estimate the probabilities of failure of an industrial system, even though they are a rare event. Instead of attack trees, they use dynamic fault trees. As we saw previously, fault trees are used in reliability and safety analysis by showing how component failures can combine to cause system failures. Dynamic fault trees, used in this work, are an extension of the standard fault trees and include additional information such as different repair policies with periodic inspections, periodic replacements, ... and allow to model common patterns in dependability models. Unlike our work, probabilities are included in the dynamic fault tree that shows the different paths that lead to a system failure. Once the tree is complete, they perform statistical analysis using the importance sampling algorithm. Importance sampling, operates a *change of measure* that modifies the probability distribution of the random variables in the model in order to make the target event happen more frequently. The final probabilities are computed using a *likelihood ratio*, that keeps track of the error by being updated every time the simulator draws a sample from a random variable. The fact that this work uses a tree structure to model issues, together with a rare event analysis matches our approach. But as application areas differ, so do methodologies. The use of dynamic fault trees is adapted to the industrial domain but could not be used in the security area. As for the sampling method, the difficulty is to find a good change of measure, and it requires to have an idea of the statistical results in order to adjust the change of measures. In our case, we don't know in advance what probabilities we are supposed to have. Another main difference is that we completely separate the system model and the attack representation. So for a given model, we can use different attack trees that represent various attacks. In this work, a fault tree represents all the possible failures of one system.

Chapter 3

Security Oriented Modeling Language

This chapter aims to introduce the IoT Security Oriented Modeling Language (IoT SOML). Here we describe the language from an abstract point of view.

This chapter is built as follow: first we introduce in Section 3.1 a simple use case that will be used throughout this part as a running example. It will allow us to illustrate the concepts we introduce in this chapter in a more concrete manner. Then, we present the IoT SOML language. For clarity reasons we chose to divide it in two parts. The first part, in Section 3.2, describes the language, leaving apart its probabilistic aspect. We give an overview of the language and its general concepts before going into the details of its abstract syntax and operational semantics. Section 3.3 integrates probabilities to the language, as well as some new concepts that become necessary in order to perform probabilistic analysis on an IoT model. We present the updated abstract syntax, the concrete syntax and operational semantics of the language and illustrate them using the running example.

IoT SOML is able to describe an IoT model, and it also includes its interactions with malicious persona performing an attack on it. In order to better describe the attack that can take place within the IoT model, we step aside from the model and delve into the details of the attack description in Section 3.4. We start by introducing the attack tree via its graphical form, and we give an equivalent formal representation.

3.1 Running Example

In this section we introduce an example involving a system and an attacker. This simple model is used as a running example throughout this chapter. It will be used to illustrate the concepts of the IoT Security Oriented Modeling Language (IoT SOML), as well as how an attack against the system can be carried out. The aim of this example is to show the nominal behaviour of a system in Figure 3.1. Then we introduce the running example represented in Figure 3.2, by modeling a successful attack performed on the system by an attacker.

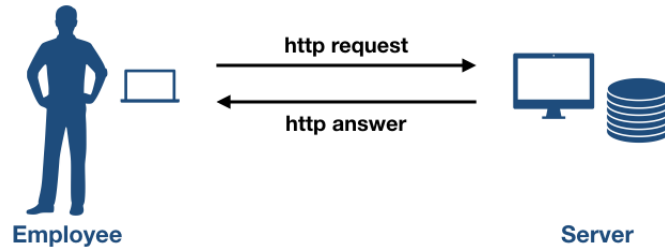


FIGURE 3.1: System Example: Nominal Behaviour

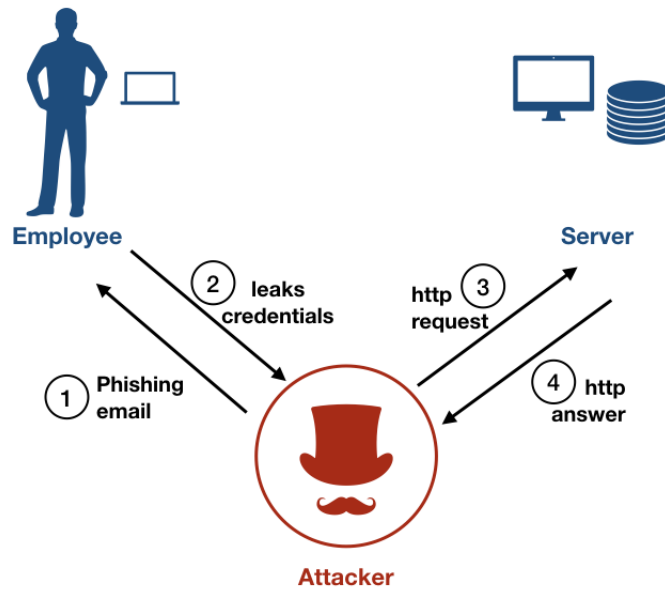


FIGURE 3.2: Running Example: Successful Attack

We consider an IoT system composed of an employee and a server. The employee has the server password and therefore can communicate with it via http requests. The nominal behaviour of this system is shown in Figure 3.1 and is composed of a communication between the employee and the server. It consists of an exchange of information between the two of them.

In the second example in Figure 3.2, we now assume that an attacker wants to gain access to the server. In the attack scenario, the attacker first sends a phishing email to the employee. The employee answers the email and by doing this, leaks his password. The attacker can now communicate with the server, using the stolen password, and therefore access the confidential information.

3.2 IoT Security Oriented Modeling Language

In this section, we introduce the IoT SOML, leaving out its probabilistic aspect. We first give an overview of the language by detailing the general concepts it contains and explaining the modeling choices we made in Subsection 3.2.1. Then, in Subsection 3.2.2, we give the language a formal perspective by defining its abstract syntax. In Subsection 3.2.3 we make preliminary explanations for the operational semantics by defining an *equivalence relation*.

Finally, in order to understand how IoT SOML behaves when it is executed we detail its operational semantics in Subsection 3.2.4 and illustrate it using the running example in Subsection 3.2.5.

3.2.1 Language General Concepts

IoT SOML describes a system as a set of *entities* that exchange information, using different means of communication. The main elements of our language are the entities: they can be either human or connected objects. In the running example we distinguish three entities: the employee, the server and the attacker. An entity has several attributes as we can see in Figure 3.3.

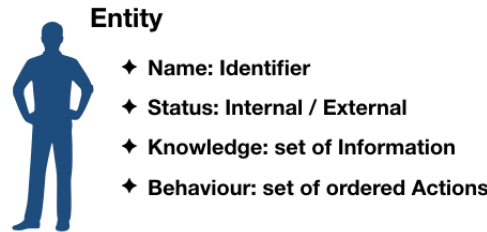


FIGURE 3.3: Entity Description

An entity has a *status* which can either be *internal* or *external*. An internal entity is part of the system, while an external entity is a malicious entity that does not belong to it. An external entity will get in contact with the system in the attempt of stealing, successfully or not, sensitive information. In the running example, the internal entities are the employee and the server. The attacker is external to the system and communicates with the employee for the sole purpose of gaining unauthorized access to the server.

An entity also possesses a number of information. This information represents the *knowledge* of the entity. This data can be either part of its original knowledge: the entity knows them from the start. For example, when the employee is created he already knows his password and his email address. But the knowledge is not static and can be expanded as additional information are acquired through the exchanges between the entities. This allows us to model the gains of access of the entities whether they are internal or external. For instance, the attacker only knows the email of the employee at the start. But as he communicates with internal entities, the attacker expands his knowledge and the new information he acquires provides him new access. For instance, we can see that through the leak of the employee's credentials, the attacker is then allowed to communicate with the server and can possibly access confidential information of the server.

Each data of the system has a *category*. In the running example we have four data

categories: email address, credentials, message and confidential data. The assignment of a category to each data is made at the creation of the system, and it is possible for the user to create as many categories as he deems necessary.

Concerning the interactions within the system: they can only happen between entities. Each communication is based on a *protocol*. Protocols integrate a verification on a certain category of data. For instance, the email protocol verification is based on the email addresses. So, for a communication based on the mail protocol to be possible, both entities must possess the email address of the receiver as part of their knowledge. This is a strong requirement and if this condition is not met, the entities cannot communicate.

Through the different interactions existing between entities, we specify a *behaviour* for each of them. The behaviour is defined as a set of ordered actions. There are 5 types of actions: Send, Receive, Leak, Collect and Internal. The Send/Receive couple represents the usual communication between two entities when they exchange information and is supported by a protocol. The receiver extends his knowledge with the data he receives. Informally, the actions consists of the following tuples:

Send : {Sender, Receiver, Protocol, Data}

Receive : {Sender, Receiver, Protocol}

The Leak/Collect actions happen between an internal and an external entity. They represent an external entity gaining access to information it is not suppose to. For example, when the employee clicks a link on the phishing email, he leaks his password, which is added to the attacker's knowledge. This communication does not go through a protocol. We express it this way:

Leak : {Sender, Receiver, Data}

Collect : {Sender, Receiver}

We decided to differentiate information obtained "legally" and information leaked to the Attacker for several reasons. First we made the choice to make some vulnerabilities of the system explicit. Secondly, the leak/collect type of verification is not based on a protocol and therefore does not include any verification. This has consequences for the attacker or other external entities who are always able to collect information as this type of communication cannot be blocked.

Finally, the internal action represents a lack of communication, it can represent an internal action that an entity does without the participation of other entities:

Internal : {}

For example, if the employee ignores the phishing email, we represent it using an internal action.

3.2.2 Abstract Syntax

The model is composed of a set of entities E that have a unique identifier. We use e_1, e_2, \dots to denote the different identifiers. The entities communicate with each others, sometimes using a set of protocols C . We use c_1, c_2, \dots to designate them. Let Val be a set of *values*, described by v_1, v_2, \dots that can be exchanged within the system. The behaviour of an entity is modeled by a CCS-like process [49]. The syntax of the process is given in Figure 3.4.

The language includes 0, that performs no action. The $a.P$ performs an action a and continues as P . $a.P + b.Q$ behaves as either $a.P$ or as $b.Q$. We use it to model

<i>Process</i>	$P, Q ::= 0 \mid a.P \mid a.P + b.Q \mid A$	
<i>Action</i>	$a, b ::= e_1 \xrightarrow[c_1]{v_1} e_2$	<i>Send</i>
	$ e_2 \xleftarrow{c_1} e_1$	<i>Receive</i>
	$ e_1 \xrightarrow[v_1]{} e_2$	<i>Leak</i>
	$ e_2 \leftarrow e_1$	<i>Collect</i>
	$ \tau$	<i>Internal</i>
<i>Definition</i>	$A \stackrel{\text{def}}{=} P$	

FIGURE 3.4: Syntax of the core IoT-calculus

a choice, so it is up to the system to decide.

We use *guarded sum* here. This means that the process is always preceded by an action: we write $a.P + b.Q$ instead of $P + Q$. This implies that we can't write $0 + Q$, however we can express $a.0 + b.Q$ or just $b.Q$. This notation is motivated by the fact that as future work, the language will be augmented with probabilities, similarly to [25]. A is a definition of a (potentially recursive) process. In Figure 3.5 we can see an example of a recursive process.

$$\begin{aligned} A &= a.A + b.B \\ B &= b.A \end{aligned}$$

FIGURE 3.5: Example of a recursive process

Entities communicate using the *Send*, *Receive*, *Leak* and *Collect* actions. During a communication entities can exchange values. In order to add the received values under the right protocol in the receiver's knowledge, we define a function

$$\text{protocol} : \text{Val} \rightarrow C$$

An entity e_1 can send to another entity e_2 a value v_1 using the protocol c_1 . The entity e_2 which receives a value has only to specify the identity of the participants, e_1 and e_2 , and the protocol c_1 . A leak action is another type of send, where the participants do not need to agree on a protocol. Finally, a collect is the counterpart of receiving in the case of a leak.

τ is an internal action, which is useful in our language for the sum construct. When an entity has a choice in the action to perform, as in $a.P + \tau.Q$, it either can do the action a or not. In the latter, the internal τ action is performed instead. However, nothing prevents a user to write a process of the form $\tau.P$. This can represent, for instance, an internal action that is not relevant for our model and it can be a way to abstract these kinds of actions.

All interactions contain two entities: the sender and the receiver. This allows us to check whether the receiver of the message is the one that was intended.

Entities' knowledge is not fixed, it can expand through the interactions of the actors of the system. A *knowledge* function associates to each entity and each protocol a subset of values, $K : E \times C \rightarrow \mathcal{P}(\text{Val})$. For simplicity we write k_i for the function $K(e_i) : C \rightarrow \mathcal{P}(\text{Val})$ and k_i^c for the set of values "known" by entity i under protocol

c.

3.2.3 Preliminaries for Operational Semantics

Here we describe some notions that will help us later define the operational semantics.

We define an **equivalence relation on processes**, denoted $\equiv_P \subseteq P \times P$ as the smallest equivalence relation which includes:

- commutativity and associativity laws for $+$:

$$\begin{aligned} P + Q &\equiv_P Q + P \\ (P + Q) + R &\equiv_P P + (Q + R) \end{aligned}$$

- the unfolding law: $A \equiv_P P$ if $A \stackrel{\text{def}}{=} P$
- and is a congruence

$$P \equiv_P Q \iff P + R \equiv_P Q + R$$

Each entity has, at any state of its computation, a running process and a knowledge. The (global) state of the system consists of the parallel composition of all entities states i.e. $s_1 \mid \dots \mid s_n$, where s_i is the current state of the entity e_i . Formally, the grammar of states is:

$$s ::= \emptyset \mid \langle P, k \rangle \mid s \mid s.$$

We also introduce a **congruence relation on states** $\equiv_s \subseteq s \times s$, as the smallest equivalence relation which:

- inherits the congruence on processes:

$$P \equiv_P Q \iff \langle P, k \rangle \equiv_s \langle Q, k \rangle$$

- includes the commutativity, associativity and the identity laws for \mid :

$$\begin{aligned} s \mid t &\equiv_s t \mid s \\ (s \mid t) \mid q &\equiv_s s \mid (t \mid q) \\ s \mid \emptyset &\equiv_s s \end{aligned}$$

- and is a congruence

$$t \equiv_s t' \iff s \mid t \equiv_s s \mid t'$$

3.2.4 Operational Semantics

The operational semantics of the language explains how the formal language that we have defined previously will be executed.

The operational semantics of the system is defined by the inference rules of Figure 3.6. The system can execute a *SendReceive* interaction between two entities e_1 and e_2 if they agree on the protocol c_1 and if they satisfy the constraint imposed by the protocol: the sender has a value in its knowledge that is also known by the receiver. The

values received by entity e_2 is added to its knowledge under the protocol c_1 . The interaction *LeakCollect* is similar, except that there are no protocols or constraints to be checked. Note that both the identity of the sender and the receiver are specified in an interaction.

The rules *Internal* and *Sum* allow to do an internal action and make a non deterministic choice, respectively.

$$\begin{array}{c}
\text{SENDRECEIVE} \frac{\langle e_1 \xrightarrow[c_{v'}]{c} e_2.P_1, k_1 \rangle | \langle e_2 \xleftarrow{c} e_1.P_2, k_2 \rangle}{\rightarrow \langle P_1, k_1 \rangle | \langle P_2, k_2^{c'} \uplus \{v'\} \rangle} \quad \exists v \in k_1^c \text{ s.t. } v \in k_2^c, c' = \text{protocol}(v') \\
\\
\text{LEAKCOLLECT} \frac{\langle e_1 \xrightarrow[c_{v'}]{c} e_2.P_1, k_1 \rangle | \langle e_2 \xleftarrow{c} e_1.P_2, k_2 \rangle}{\rightarrow \langle P_1, k_1 \rangle | \langle P_2, k_2^{c'} \uplus \{v'\} \rangle} \quad c' = \text{protocol}(v') \\
\\
\text{INTERNAL} \frac{}{\langle \tau.P, k \rangle \rightarrow \langle P, k \rangle} \\
\\
\text{SUM} \frac{\langle P_i, k_i \rangle | \langle P_j, k_j \rangle \rightarrow \langle P'_i, k'_i \rangle | \langle P'_j, k'_j \rangle}{\langle P_i + Q_i, k_i \rangle | \langle P_j + Q_j, k_j \rangle \rightarrow \langle P'_i, k'_i \rangle | \langle P'_j, k'_j \rangle} \\
\\
\text{PAR} \frac{s \rightarrow s'}{s \mid t \rightarrow s' \mid t} \\
\\
\text{CONGRUENCE} \frac{s \equiv_s t \rightarrow s' \equiv_s t'}{s \rightarrow s'}
\end{array}$$

FIGURE 3.6: The operational semantics of an IoT system

The two remaining rules *Par* and *Congruence* are on states and ensure that the rules can proceed regardless of the syntactical form of the system.

3.2.5 Operational Semantics Application on Running Example

Let's see now the operational semantics of our running example.

Example 1. Let $E = \{\text{attacker}, \text{server}, \text{employee}\}$ be three entities which communicate with each other using five protocols $C = \{\text{http}, \text{mail}, \text{message}, \text{credentials}, \text{confidentialData}\}$. The *employee* has access to the confidential data stored in the server thanks to his credentials, using the *http* protocol. The *attacker* wants to get access to these confidential data, therefore he must first steal the *employee*'s credentials (as we saw in Figure 3.2).

- **Employee :**

At the beginning the *employee* knows his email address and his credentials to access the server information. Whoever has his email address can communicate with him.

Concerning his behaviour, he receives an email from the *attacker*. He then has

the choice between leaking his credentials or ignoring the attacker's email. Below is a formal representation of the employee's behaviour and knowledge:

$$\begin{aligned}
 \text{Employee} &= \text{employee} \xleftarrow{\text{mail}} \text{attacker}.(\text{employee} \xrightarrow[\text{employeeCredentials}]{} \text{attacker}.\text{Employee} \\
 &\quad + \tau.\text{Employee}) \\
 k_{\text{employee}} &= \{ \text{mail} = \{\text{employeeEmail}\}, \\
 &\quad \text{message} = \emptyset, \\
 &\quad \text{credentials} = \{\text{employeeCredentials}\}, \\
 &\quad \text{confidentialData} = \emptyset, \\
 &\quad \text{http} = \emptyset \}
 \end{aligned}$$

- **Server :**

The initial knowledge of the server is composed of the confidential information and of a set of credentials. The credentials belong to the entities with authorized access to the confidential information.

If the *attacker* manages to steal some credentials he will send a request to the server. Following this request the *server* can either send him the confidential information or ignore the request.

Formally, it becomes:

$$\begin{aligned}
 \text{Server} &= \text{server} \xleftarrow{\text{http}} \text{attacker}.(\text{server} \xrightarrow[\text{secretInformation}]{} \text{attacker}.\text{Server} + \tau.\text{Server}) \\
 k_{\text{server}} &= \{ \text{mail} = \emptyset, \\
 &\quad \text{message} = \emptyset, \\
 &\quad \text{credentials} = \{\text{employeeCredentials}\}, \\
 &\quad \text{confidentialData} = \{\text{secretInformation}\} \\
 &\quad \text{http} = \emptyset \}
 \end{aligned}$$

- **Attacker :**

Initially, the *attacker*'s knowledge is composed of the employee's email and of different messages he plans on sending the other entities (for instance the content of the email he sends to the *employee*).

We define his behaviour the following way: he can choose to either send an email to the *employee* or a http request to the *server*.

Here is how we formally represent the *attacker*'s behaviour and knowledge:

$$\begin{aligned}
\text{Attacker} &= \text{attacker} \xrightarrow[\text{giveCredentials}]{\text{mail}} \text{employee.Attacker} + \\
&\quad \text{attacker} \xrightarrow[\text{giveConfidentialData}]{\text{http}} \text{server.Attacker} + \\
&\quad \text{attacker} \leftarrow \text{employee.Attacker} + \text{attacker} \leftarrow \text{server.Attacker} \\
k_{\text{attacker}} &= \{ \text{mail} = \{ \text{employeeEmail} \}, \\
&\quad \text{message} = \{ \text{giveCredentials}, \text{giveConfidentialData} \}, \\
&\quad \text{credentials} = \emptyset, \\
&\quad \text{confidentialData} = \emptyset \\
&\quad \text{http} = \emptyset \}
\end{aligned}$$

Let's see now how we represent a successful attack towards the system.

1. First, the attacker sends an email to the employee:

$$\begin{aligned}
&\langle \text{Attacker}, k_{\text{attacker}} \rangle \mid \langle \text{Employee}, k_{\text{employee}} \rangle \rightarrow \\
&\quad \langle \text{Attacker}, k_{\text{attacker}} \rangle \mid \langle P_E, k'_{\text{employee}} \rangle
\end{aligned}$$

where

$$P_E = \text{employee} \xrightarrow[\text{employeeCredentials}]{} \text{attacker.Employee} + \tau.\text{Employee}$$

and where the knowledge of the employee changes:

$$\begin{aligned}
k'_{\text{employee}} &= \text{mail} = \{ \text{employeeEmail} \}, \\
&\quad \text{message} = \{ \text{giveCredentials} \}, \\
&\quad \text{credentials} = \{ \text{employeeCredentials} \}, \\
&\quad \text{confidentialData} = \emptyset \\
&\quad \text{http} = \emptyset \}
\end{aligned}$$

2. Once he receives the attacker's email, the employee has two options:

2. a) He can leak his credentials to the attacker:

$$\begin{aligned}
&\langle \text{Attacker}, k_{\text{attacker}} \rangle \mid \langle P_E, k'_{\text{employee}} \rangle \rightarrow \\
&\quad \langle \text{Attacker}, k'_{\text{attacker}} \rangle \mid \langle \text{Employee}, k'_{\text{employee}} \rangle
\end{aligned}$$

In this case, the attacker's knowledge is changed once he acquires the credentials :

$$\begin{aligned} k'_{attacker} = & \{mail = \{employeeEmail\}, \\ & message = \{giveCredentials, giveConfidentialData\}, \\ & credentials = \{employeeCredentials\}, \\ & confidentialData = \emptyset \\ & http = \emptyset\} \end{aligned}$$

2. b) or ignore the email:

$$\begin{aligned} & \langle Attacker, k_{attacker} \rangle \mid \langle P_E, k'_{employee} \rangle \rightarrow \\ & \langle Attacker, k_{attacker} \rangle \mid \langle Employee, k'_{employee} \rangle \end{aligned}$$

Here, the knowledge of the Attacker remains the same

3. Now the attacker decides to interact with the Server

3. a) Following 2.a, if the Attacker managed to steal the Employee's credentials, he can successfully communicate with the server. The attacker sends a http request to the server in order to access the confidential information.

$$\begin{aligned} & \langle Attacker, k'_{attacker} \rangle \mid \langle Server, k_{server} \rangle \rightarrow \\ & \langle Attacker, k'_{attacker} \rangle \mid \langle P_S, k'_{server} \rangle \end{aligned}$$

where $P_S = server \xrightarrow[\text{secretInformation}]{\quad} attacker.Server + \tau.Server$
and where the knowledge of the Server changes:

$$\begin{aligned} k'_{server} = & \{mail = \emptyset, \\ & message = \{giveConfidentialData\}, \\ & credentials = \{employeeCredentials\}, \\ & confidentialData = \{secretInformation\} \\ & http = \emptyset\} \end{aligned}$$

3. b) Following 2.b, the Attacker doesn't have the Employee's credentials. The Attacker doesn't have the mandatory information for a http communication with the Server. The scenario stops here and the Attacker goes back to his initial state, with his knowledge remaining unchanged.

4. Following 3.a, the Server has received the Attacker's request and the Server can leak the confidential information to the Attacker:

$$\begin{aligned} & \langle Attacker, k'_{attacker} \rangle \mid \langle P_S, k'_{server} \rangle \rightarrow \\ & \langle Attacker, k''_{attacker} \rangle \mid \langle Server, k'_{server} \rangle \end{aligned}$$

In this case, the Attacker's knowledge evolves and includes the secret information of the Server:

$$\begin{aligned} k''_{attacker} = & \{mail = \{employeeEmail\}, \\ & message = \{giveCredentials, giveConfidentialData\}, \\ & credentials = \{employeeCredentials\}, \\ & confidentialData = \{secretInformation\} \\ & http = \emptyset\} \end{aligned}$$

or refuse to leak them:

$$\begin{aligned} & \langle Attacker, k'_{attacker} \rangle \mid \langle P_S, k'_{server} \rangle \rightarrow \\ & \langle Attacker, k'_{attacker} \rangle \mid \langle Server, k'_{server} \rangle \end{aligned}$$

In this case, the knowledge of the Attacker remains unchanged.

From the moment the attacker has the confidential information in his knowledge, we consider the attack to be a success.

Another way for the execution to stop is with a *deadlock*. To better understand this, let's consider an alternative scenario, where the behaviour of the Attacker is slightly different:

$$\begin{aligned} Attacker = & attacker \xrightarrow[\text{giveCredentials}]{mail} employee.Attacker + \\ & \tau.attacker \xrightarrow[\text{giveConfidentialData}]{http} server.Attacker + \\ & attacker \leftarrow employee.Attacker + attacker \leftarrow server.Attacker \\ k_{attacker} = & \{mail = \{employeeEmail\}, \\ & message = \{giveCredentials, giveConfidentialData\}, \\ & credentials = \emptyset, \\ & confidentialData = \emptyset \\ & http = \emptyset\} \end{aligned}$$

In this new behaviour, the action of sending a request to the Server is preceded by an internal action τ .

Consider now the section 3.b of the previous scenario: the Attacker wants to send a request to the Server without having the credentials. The Attacker starts by performing the action τ . The next possibility for him is $attacker \xrightarrow[\text{giveConfidentialData}]{http} server.Attacker$. But this is not an option as this communication is forbidden by the protocol verification. Therefore, the Attacker is stuck in a behaviour: he can't perform his only possible action. The whole system is stuck in a state and has no other option than to stop. The execution's interruption is reported by a deadlock.

Going back to the part 2.a of the attack scenario, we formally explain the behavior of the Employee in Figure 3.7.

$$\begin{array}{c}
\frac{}{\langle employee \xrightarrow{employeeCredentials} attacker.E + \tau.E, k_e \rangle | \langle attacker \leftarrow employee.A + \dots, k_a \rangle} \\
\frac{}{\langle e_1 + e_2, k_{employee} \rangle | A \equiv_s \langle e_2 + e_1, k_e \rangle | A \rightarrow \langle E, k_e \rangle | \langle A, k_a \rangle} \\
\frac{}{\langle e_1, k_e \rangle | \langle attacker \leftarrow employee.A, k_a \rangle \rightarrow \langle E, k_e \rangle | \langle A, k'_a \rangle} \\
\hline
\frac{}{\langle e_1 + e_2 \equiv_p e_2 + e_1 \rangle} \text{COMMUTATIVITY} \\
\frac{}{\langle e_1 + e_2, k_e \rangle \equiv_s \langle e_2 + e_1, k_e \rangle} \text{CONGRUENCE} \\
\frac{}{\text{SUM}} \text{CONGRUENCE}
\end{array}$$

where $e_1 = employee \xrightarrow{employeeCredentials} attacker.E$

and $e_2 = \tau.E$

and where the full behavior of the *attacker* can be written

$$A = attacker \xrightarrow{mail} employee.A + attacker \xrightarrow{giveSecretInformation} employee.A + employee \leftarrow attacker.A + server \leftarrow attacker.A.$$

But for clarity reasons, we shorten it to :

$A = employee \leftarrow attacker.A + \dots$ We also apply the Commutativity and Congruence rules on the *attacker's* behaviour in order for its actions to be in the right order. But we don't write it explicitly as we did for the *employee* behavior for the figure to be clearer.

FIGURE 3.7: Formal description of Employee behaviour

3.3 Probabilistic IoT Security Oriented Modeling Language

IoT SOML alongside with Statistical Model Checking are used to determine the chances of a given attack to succeed on a system. This requires to integrate probabilities on the actions of the entities. We present the extended version of the abstract syntax in Subsection 3.3.1 and operational semantics as well as new concepts needed to handle the probabilities in Subsection 3.3.2. Then, in Subsection 3.3.3 we apply the new operational semantics on the running example to illustrate the changes made in the language. Finally, in Subsection 3.3.4 presents the concrete syntax and describes how we defined the grammar of the IoT Modeling Language, based on the formal rules we set in the previous subsections.

3.3.1 Extended Abstract Syntax

As in section 3.2, each entity has a unique identifier, denoted by e_1, \dots, e_n and a running process. The grammar of processes is defined in Figure 3.8.

We write C for a set of protocols, ranged over by c and Val for a set of values ranged over by v . We handle the knowledge of the entities, protocols and their inherent verifications the way we did before. The actions of a process remain unchanged and we still distinguish between "safe" interactions consisting in sending and receiving values and the ones that can potentially lead to security issues.

A new concept is expressed in the grammar: the *threads*. Processes are composed of threads, which can only do sequential computations. In the former definition, the processes were composed of only one thread. This means that when an entity engaged in a behaviour, it had to execute all the successive actions in the order defined by the behaviour, and it was a blocking process. They could not start the behaviour and move on to another one before it was completed. Now, the entities can engage in several parallel set of actions. In addition, when an entity has several options about the actions to perform, the choice was previously made internally at the runtime. Now this choice is made explicitly through the choice function, that calculates a random number.

We write 0 for the inactive process and A for the (recursive) definitions of threads. Actions are equipped with a *probability*, denoted by $[n]a$, for an action a and a probability $n \in [0, 1]$. Threads can therefore do a probabilistic choice between actions, with the restriction that the sum of the probabilities of all available actions is 1. If there is only one available action, its probability is 1 and can be omitted.

3.3.2 Operational Semantics

Each entity has, at any state of its computation, a running process P and a knowledge k . The global state of the system consists of the parallel composition of all entities states $s_1 \mid \dots \mid s_n$, where s_i is the current state of the entity e_i .

In this semantics, a probabilistic choice is always resolved locally, using the CHOICE rule. A transition derived by the CHOICE rule is considered internal and is labeled τ . A process can also do internal transitions using the INTERNAL rule.

The semantics of the probabilistic language includes new elements. First, transitions now include two labels: a probabilistic one that provides the chances for the action to happen and another one that indicates whether the transition is a Send/Receive or a Leak/Collect.

Then, there is a CONGRUENCE rule that extends the congruence relation of Section

$$\begin{array}{ll}
\text{Process } P, Q ::= T \mid P \mid Q & \\
\text{Thread } T, U ::= 0 \mid A \mid \sum_{i \in I} [n_i] a_i. T_i \text{ where } n_i \in (0, 1], & \\
\sum_{i \in I} n_i = 1 & \\
\text{Action } a, b ::= e \xrightarrow[v]{c} e' & \text{Send} \\
| e \xleftarrow{c} e' & \text{Receive} \\
| e \xrightarrow[v]{} e' & \text{Leak} \\
| e \leftarrow e' & \text{Collect} \\
| \tau & \text{Internal} \\
\text{Definition } A \stackrel{\text{def}}{=} T & \\
\text{State } s ::= \emptyset \mid \langle P, k \rangle \mid s \mid s. &
\end{array}$$

FIGURE 3.8: Syntax of the probabilistic IoT-calculus

3.2, by also including the associativity and the commutativity for \mid , the identity element \emptyset for \mid .

The congruence relation on the states remains unchanged.

Another new rule is the PARPROC rule that allows one to use congruence and parallel composition on processes to derive transitions.

And finally the rules for the global states, PARSTATE, shows how we choose a global transition from several local ones using an uniform distribution. We rely on two auxiliary functions, count_τ and $\text{count}_{\text{SR}, \text{LC}}$ that count the number of local transitions with labels τ and labels SR (Send/Receive), LC (Leak Collect), respectively. A probabilistic choice is made between all available transitions.

Definition 1 (Counting τ transitions from a state). The functions $\text{count}_\tau : \text{State} \rightarrow \mathbb{N}$ and $\text{count_proc}_\tau : \text{Proc} \rightarrow \mathbb{N}$ are defined as follows:

$$\begin{aligned}
\text{count}_\tau(s|t) &= \text{count}_\tau(s) + \text{count}_\tau(t) \\
\text{count}_\tau(\langle P, k \rangle) &= \text{count_proc}_\tau(P) \\
\text{count_proc}_\tau(0) &= 0 \\
\text{count_proc}_\tau(\alpha.P) &= 0 \text{ if } \alpha \neq \tau \\
&= 1 \text{ if } \alpha = \tau \\
\text{count_proc}_\tau(\sum \alpha_i.P_i) &= \sum \text{count_proc}_\tau(\alpha_i.P_i) \\
\text{count_proc}_\tau(P \mid Q) &= \text{count_proc}_\tau(P) + \text{count_proc}_\tau(Q).
\end{aligned}$$

For counting the number of interactions, we have first to rewrite a state such that it is always in the following canonical form:

$$s \equiv s_S \mid s_R \mid s_L \mid s_C \quad \text{where} \quad
\begin{aligned}
s_S &= \langle P_1^S, k_1^S \rangle \mid \dots \mid \langle P_{n_S}^S, k_{n_S}^S \rangle \\
s_R &= \langle P_1^R, k_1^R \rangle \mid \dots \mid \langle P_{n_R}^R, k_{n_R}^R \rangle \\
s_L &= \langle P_1^L, k_1^L \rangle \mid \dots \mid \langle P_{n_L}^L, k_{n_L}^L \rangle \\
s_C &= \langle P_1^C, k_1^C \rangle \mid \dots \mid \langle P_{n_C}^C, k_{n_C}^C \rangle
\end{aligned}$$

$$\begin{array}{c}
\text{CHOICE} \frac{}{\langle \sum_{i \in I} [n_i] a_i.T_i, k \rangle \xrightarrow{\tau} \langle a_i.T_i, k \rangle} \\
\\
\text{INTERNAL} \frac{}{\langle \tau.P, k \rangle \xrightarrow{\tau} \langle P, k \rangle} \\
\\
\text{SENDRECEIVE} \frac{}{\langle e_1 \xrightarrow[v']{c} e_2.P_1, k_1 \rangle | \langle e_2 \xleftarrow{c} e_1.P_2, k_2 \rangle \xrightarrow[\text{SR}:v']{[1]} \langle P_1, k_1 \rangle | \langle P_2, k_2^{c'} \uplus \{v'\} \rangle} \exists v \in k_1^c \text{ s.t. } v \in k_2^c, c' = \text{protocol}(v') \\
\\
\text{LEAKCOLLECT} \frac{}{\langle e_1 \xrightarrow[v']{} e_2.P_1, k_1 \rangle | \langle e_2 \xleftarrow{} e_1.P_2, k_2 \rangle \xrightarrow[\text{LC}:v']{[1]} \langle P_1, k_1 \rangle | \langle P_2, k_2^{c'} \uplus \{v'\} \rangle} c' = \text{protocol}(v') \\
\\
\text{PARPROC} \frac{\langle P_i, k_i \rangle | \langle P_j, k_j \rangle \xrightarrow[l]{[n]} \langle P'_i, k'_i \rangle | \langle P'_j, k'_j \rangle}{\langle P_i \mid Q_i, k_i \rangle | \langle P_j \mid Q_j, k_j \rangle \xrightarrow[l]{[n]} \langle P'_i \mid Q_i, k'_i \rangle | \langle P'_j \mid Q_j, k'_j \rangle} \\
\\
\text{CONGRUENCE} \frac{s \equiv_s t \xrightarrow[l]{[n]} s' \equiv_s t'}{s \xrightarrow[l]{[n]} s'} \\
\\
\text{PARSTATE} \frac{s \xrightarrow[l]{[n]} s'}{s|t \xrightarrow[l]{[1/m, n]} s'|t} m = \text{count}_\tau(s|t) + \text{count}_{\text{SR}, \text{LC}}(s|t)
\end{array}$$

FIGURE 3.9: The operational semantics of a probabilistic IoT system

and where $P_i^S \equiv a.P$ and the action a is a send; nS is the number of processes of the form above in s and $i \in [1; nS]$. Similarly we define the rest of the processes. Note that if we cannot rewrite a state in this form then the rule PARSTATE cannot be applied. Moreover entities can only communicate with other entities, that is interactions are not defined internally to an entity. We therefore only need to count interactions between entities.

The function $\text{count}_{\text{SR}, \text{LC}}$ uses an auxiliary function $\bar{\cdot} : \text{action} \rightarrow \text{action}$ which defines an action \bar{a} which can synchronise with a using the rules SENDRECEIVE or LEAKCOLLECT.

Definition 2. Let $s \equiv s_S \mid s_R \mid s_L \mid s_C$ be a state in a canonical form. The function $\text{count}_{\text{SR}, \text{LC}} : \text{State} \rightarrow \mathbb{N}$ is defined on s as follows:

$$\begin{aligned} \text{count}_{\text{SR}, \text{LC}}(s_S \mid s_R \mid s_L \mid s_C) &= \text{count}_{\text{SR}}(s_S, s_R) + \text{count}_{\text{LC}}(s_L \mid s_C) \\ \text{count}_{\text{SR}}(\langle a.P, k \rangle \mid s, t) &= \text{count}(a, t) + \text{count}_{\text{SR}}(s, t) \\ \text{count}_{\text{LC}}(\langle a.P, k \rangle \mid s, t) &= \text{count}(a, t) + \text{count}_{\text{LC}}(s, t) \\ \text{count}(a, \langle b.P, k \rangle \mid t) &= 1 + \text{count}(a, t) \text{ if } a = \bar{b} \\ &= \text{count}(a, t) \text{ otherwise} \end{aligned}$$

The operational semantics of Figure 3.9, defines a transition system (S, T, L, s_0) where we write S for a set of states, ranged over by s with s_0 the initial state, $L \subseteq \{\tau\} \cup (\{\text{SR}, \text{LC}\} \times \text{Val})$ for a set of labels, ranged over by l , and $T \subseteq S \times [0, 1] \times L \times S$ for a set of transitions, where each transition is decorated by a probability and by a label.

3.3.3 Operational Semantics Application on Running Example

To illustrate the new operational semantics, we use the first part of the running example we defined earlier (see Section 3.1). We consider only the first step of the attack: the attacker sends an email to the employee, and the employee can either leak his credentials, or ignore the email. Using the new operational semantics we can assign probabilities to the employee's actions.

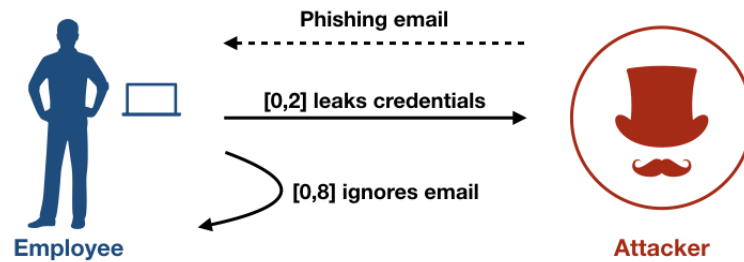


FIGURE 3.10: Probabilistic Example

Figure 3.10 shows the first part of the running example. Once the attacker has reached to the employee, the employee can leak or ignore the email. We consider the chances for the employee to ignore the email to be greater so we assigned a probability of 0.8 compared to 0.2 for the leak of his credentials.

The following example is an IoT trace for the example we described earlier in Example 3.10. This trace shows the *attacker* sending an email to the *employee*, and the *employee* leaking his credentials.

Example 2. We define E the behaviour of the Employee, S the behaviour of the Server and A the behaviour of the Attacker by:

$$\begin{aligned}
 E &= \underbrace{\text{employee} \xleftarrow{\text{mail}} \text{attacker}}_{e_1} . ([n_1]\tau.E + [n_2] \underbrace{\text{employee} \xrightarrow{\text{employeeCredentials}} \text{attacker}.E}_{e_2}) \\
 A &= [n_3] \underbrace{\text{attacker} \xrightarrow[\text{giveCredentials}]{\text{mail}} \text{employee}.A}_{a_1} + [n_4] \underbrace{\text{attacker} \xrightarrow[\text{giveSecretInformation}]{\text{http}} \text{server}.A}_{a_2} \\
 &\quad + [n_5] \underbrace{\text{attacker} \leftarrow \text{employee}.A}_{a_3} + [n_6] \underbrace{\text{attacker} \leftarrow \text{server}.A}_{a_4}
 \end{aligned}$$

The execution gives us the following trace:

$$\begin{aligned}
 &\langle e_1.([n_1]\tau.E + [n_2]e_2.E), k_e \rangle | \langle [n_3]a_1.A + [n_4]a_2.A + [n_5]a_3.A + [n_6]a_4.A, k_a \rangle \xrightarrow[\tau]{[n_3]} \\
 &\langle e_1.([n_1]\tau.E + [n_2]e_2.E), k_e \rangle | \langle a_1.A, k_a \rangle \xrightarrow[\text{SR:giveCredentials}]{[1]} \\
 &\langle [n_1]\tau.E + [n_2]e_2.E, k_e^{\text{mail}} \uplus \{\text{giveCredentials}\} \rangle | \langle A, k_a \rangle \equiv_s \\
 &\langle [n_1]\tau.E + [n_2]e_2.E, k_e \rangle | \langle [n_3]a_1.A + [n_4]a_2.A + [n_5]a_3.A + [n_6]a_4.A, k_a \rangle \xrightarrow[\tau]{[n_2]} \\
 &\langle e_2.E, k_e \rangle | \langle [n_3]a_1.A + [n_4]a_2.A + [n_5]a_3.A + [n_6]a_4.A, k_a \rangle \xrightarrow[\tau]{[n_5]} \\
 &\langle e_2.E, k_e \rangle | \langle a_3.A, k_a \rangle \xrightarrow[\text{LC:employeeCredentials}]{[1]} \langle E, k_e \rangle | \langle A, k_a^{\text{credentials}'} \uplus \{\text{employeeCredentials}\} \rangle
 \end{aligned}$$

In the trace, we can see that a first choice is made between the *attacker*'s actions and another choice is made between *employee* $\xrightarrow[\text{employeeCredentials}]{\text{mail}} \text{attacker}$ and τ . The probability of the trace is then $n_3 \cdot 1 \cdot n_2 \cdot n_5 \cdot 1$.

Using this example, we now want to apply the count functions on the trace.

First, we need to rewrite the states for them to match the form we defined in Definitions 1 and 2.

At the initial state, we have:

$$\begin{aligned}
 s_S &= \langle A, k_a \rangle \\
 s_R &= \langle E, k_e \rangle \\
 s_L &= 0 \\
 s_C &= \langle A, k_a \rangle
 \end{aligned}$$

Applying the count function:

$$\begin{aligned}
\text{count}_{\text{SR}}(\langle A, k_a \rangle \mid \langle E, k_e \rangle) &= 1 \\
\text{count}_{\text{LC}}(0 \mid \langle A, k_a \rangle) &= 0 \\
\text{count}_{\text{SR,LC}}(s_S \mid s_R \mid s_L \mid s_C) &= 1 \\
\text{count}_{\tau}(\langle A, k_a \rangle) &= 0 \\
\text{count}_{\tau}(\langle E, k_e \rangle) &= 0
\end{aligned}$$

This means that at this stage, there is only one interaction possible: a Send/Receive communication between the *attacker* and the *employee*.

After the first step of execution, once the *attacker* has sent the email to the *employee* and the *employee* has received it:

$$\begin{aligned}
s_S &= \langle A, k_a \rangle \\
s_R &= 0 \\
s_L &= \langle E', k_e \rangle \quad \text{where } E' = \tau.E + e_2.E \\
s_C &= \langle A, k_a \rangle
\end{aligned}$$

Applying the count function:

$$\begin{aligned}
\text{count}_{\text{SR}}(\langle A, k_a \rangle \mid 0) &= 0 \\
\text{count}_{\text{LC}}(\langle E, k_e \rangle \mid \langle A, k_a \rangle) &= 1 \\
\text{count}_{\text{SR,LC}}(s_S \mid s_R \mid s_L \mid s_C) &= 1 \\
\text{count}_{\tau}(\langle A, k_a \rangle) &= 0 \\
\text{count}_{\tau}(\langle E', k_e \rangle) &= 1
\end{aligned}$$

At this point there are two possibilities: a Leak/Collect interaction between the *attacker* and the *employee*, or a τ action from the *employee*.

3.3.4 Concrete Syntaxe

We defined the grammar of the language using the extended Backus-Naur form (EBNF) notation, which is the most commonly used formalism to express a grammar. The grammar description is contained in a file, of which we can see parts in Figure 3.11, Figure 3.12 and Figure 3.13.

The grammar definition is made of two types of *rules* that express the correct syntax of the language: the *lexer rules* and the *parser rules*. The lexer rules, that we can see in Figure 3.11, specify how the basic elements of the language are represented. For example, in Figure 3.11 we define the term "Id", in the first line, as a mix of letters (upper or lower case) and digits, but it has to start with a letter (either upper or lower case). The parser rules represent elements of the language made of several other elements.

```

/***** Lexer rules *****/
Id : [a-zA-Z][0-9A-Za-z_]* ;
Name : [A-Z][A-Za-z_]* ;
Status : ( '_Internal' | '_External' ) ;
Int : [0-9]+ ;

```

FIGURE 3.11: Grammar Definition: Lexer rules

Before describing the syntax of the IoT language, we need to introduce a new concept of the language: the *data category*. As we know every actor of the system possesses a basic knowledge and can send and receive data in order to expand its knowledge. We associate a category to each data. For instance in a mail communication, both actors will interact by sending each other a message and using the email address of the receiver as verification. We associate to the value of the message a category "message" and the same goes for the email address with the category "email". According to the parser rules described in Figure 3.12, an IoT file starts with the global definition of the protocols, categories of values and values used in the system. First we list all the protocols of the system and link each of them with a category of value for verification. For example, the verification of the mail protocol is performed on the email address. As we saw previously in this chapter this means that for a communication to be possible through the mail protocol, both entities must have the email address of the receiver on their knowledge. Then we list all the categories of values existing in the system. Two examples of value categories are the "email address" and the "message" both used for an email exchange. Next, we can find all the existing values of the system and their category, for example each email address and message circulating in the system are declared here.

These global declarations of data, categories and protocol is meant to help us during the parsing process. It also makes possible for the actors not to have to indicate the category of the value they send whenever they are communicating with each other.

Then we define the entities. As shown in Figure 3.12 an entity is composed of a name that is unique and is used as an identifier. It also consists in data declarations, action declarations and a behaviour declaration. The data represents the initial knowledge of the entity, it can be used as verification in protocols and/or it can be communicated to other entities. The actions are declared here and used later to compose the behaviour of the entity. It is not possible to include in the behaviour of the entity an action that was not primarily defined here. We distinguish five types of actions, shown in Figure 3.12, with different parameters depending on their needs. For each action, except the Internal ones we stipulate the Sender and the Receiver, using the names of the entities involved.

- Send We add the protocol that supports the communication, and the value sent. The category of the value sent is not specified as it was defined previously in the global values declaration.
- Receive We only specify the protocol through which the communication is carried.
- Leak This communication is not based on any protocol so we only specify the value that is leaked.
- Collect Here too, there is no protocol needed, so we just need to know which are the entities involved in the communication.

Internal This action represents a lack of communication so there are no parameters.

Finally, we specify the entity behaviour. A behaviour is a set of ordered actions. The actions can be preceded by a weight, or not. The weight is what is later transformed in probabilities when the model is executed. One entity can possess several behaviours, but we have to precise a "main" behaviour that is the entry point.

Going back to our running example, we can see in Figure 3.13 how we define it in IoT SOML.

3.4 Attack Representation using Attack Trees

As for now we can represent an IoT system with its internal entities performing their nominal behaviour. We can also perform an attack on this system, using external entities and associate probabilities to the different actors' actions. To do so, the behaviour of the attacker and other external entities (like Malwares, infected devices, etc) are already described in the IoT model. This means that the IoT model expresses, in addition to its normal behaviour, one or several successful attacks. But, so far, nothing in our model permits to identify a successful attack, or even to be able to tell if an attack is currently in progress: this is why we need attack trees.

In Section 2.2.1, we pointed out the benefits of attack trees and explained how they can be used. Here, we first describe how an attack tree is built, using a visual example in Subsection 3.4.1. Then, in Subsection 3.4.2 we give a formal representation of the attack trees that we use in our work.

3.4.1 Attack Tree Graphical Representation

An attack tree is meant to represent an attack and the multiple steps that compose it. The ultimate goal of the attack is expressed in the root node of the tree: it describes when an attack succeeds. For instance, it can be for instance to access confidential information, steal money or data. This objective is decomposed into simple actions or steps that the Attacker must carry out in order to complete the attack.

To represent the behaviour of the Attacker we have two different types of nodes: the internal nodes and the leaves. A leaf is a node without children and an internal node has, in our case, two or more successors that can be either leaves, internal nodes or both.

The leaves represent the actions carried out by the Attacker. For example, in Figure 3.14 we have four simple actions that the Attacker can choose to perform: *a*, *b*, *c* and *d*. The internal nodes express a subgoal of the Attacker and how to achieve it. There are two types of internal nodes: the *AND* node and the *OR* node. The *AND* node means that all its children must be completed in order for the subgoal to be reached. In the example described in the Figure 3.14, *subgoal1* is completed only if actions *a* and *b* are done. In the same way, the root is reached and the attack successful only if *subgoal1* and *subgoal2* are achieved. As for an *OR* node, only one action among its successors must necessarily be executed in order to complete it. In the example of the Figure 3.14, *subgoal2* is reached if *c* or *d* are taken.

3.4.2 Attack Tree Formal Representation

Syntactic Definition of Attack Trees

Definition 3 (Attack Tree). Let $\Delta \subseteq \{\text{SR}, \text{LC}\} \times \text{Val}$ be a set of events. An *attack tree* t is a term constructed recursively from the set Δ using the operators \vee and \wedge .

We denote \mathcal{T} to be the set of attack trees.

The height and the depth of the attack tree are important notions that we need in order to perform a successful analysis of the on-going attack.

Definition 4 (Height and depth in an attack tree). The height of t , denoted $h(t)$, is defined recursively as follows:

$$h(t) = \begin{cases} 1 & \text{if } t \in \Delta \\ 1 + \max(h(t_1), h(t_2)) & \text{if } t = t_1 \wedge t_2 \text{ or } t = t_1 \vee t_2 \end{cases}$$

Each node n in a tree t has a *depth*, defined as follows:

$$d(n, t) = \begin{cases} 0 & \text{if } n = t \\ 1 + d(n, t_1) & \text{if } t = t_1 \wedge t_2 \text{ or } t = t_1 \vee t_2 \text{ and } n \in t_1 \\ 1 + d(n, t_2) & \text{if } t = t_1 \wedge t_2 \text{ or } t = t_1 \vee t_2 \text{ and } n \in t_2 \end{cases}$$

Considering the attack tree t represented in figure 3.15, we can compute its height and the depth of several nodes of its nodes.

The height of the tree t can be computed in the following way:

$$h(t) = 1 + \max(h(t_1), h(t_2))$$

We need then to calculate the depth of t_1 and t_2 . We define

$$h(t_1) = 1 + \max(1, h(t_3)) = 3$$

and

$$h(t_2) = 1 + \max(h(i), h(j)) = 2$$

We can conclude that

$$h(t) = 4$$

Using the definition above we can compute the depth of the nodes of the tree t . For instance we have

$$d(A, t) = 0$$

$$d(i, t) = 1 + d(i, t_2) = 1 + (1 + d(i, i)) = 2$$

$$d(g, t) = 1 + d(g, t_1) = 1 + (1 + d(g, t_3)) = 1 + (1 + (1 + d(g, g))) = 3$$

Semantics Definition of Attack Trees

In the following definition we introduce a Boolean variable v_e for every event $e \in \Delta$ such that $e \neq e' \iff v_e \neq v_{e'}$.

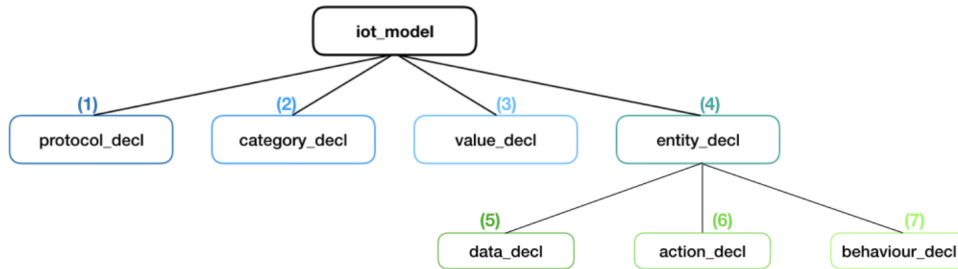
Definition 5 (Attack Tree Semantics). Let $t \in \mathcal{T}$ an attack tree. The semantics of t , denoted $\llbracket t \rrbracket$, consists of a propositional formula defined by recursion on t as follows:

- if $t \in \Delta$ then $\llbracket t \rrbracket = v_t$;
- if $t = t_1 \wedge t_2$ then $\llbracket t \rrbracket = \llbracket t_1 \rrbracket \wedge \llbracket t_2 \rrbracket$;
- if $t = t_1 \vee t_2$ then $\llbracket t \rrbracket = \llbracket t_1 \rrbracket \vee \llbracket t_2 \rrbracket$.

The semantics of an attack tree is given with respect to a valuation of the Boolean variables. Let $\mathbf{X} : \Delta \rightarrow \{\text{true}, \text{false}\}$ be a valuation for Δ , then the semantics of t

w.r.t. \mathbf{X} , denoted $\llbracket t \rrbracket(\mathbf{X}) \in \{true, false\}$, consists in evaluating the associated Boolean formula [37].

In order to assess whether an attack was successful, we monitor the executions of a system. Given an execution trace σ , its corresponding valuation $\mathbf{X}(\sigma)$ sets v_e to *true* if the event e occurred in σ . If $\llbracket t \rrbracket(\mathbf{X}(\sigma))$ is true then the execution σ is a successful attack of t .



```

protocol_decl : 'Protocol' protocol_name 'checks' category_name ; (1)
category_decl : 'ValueCategory' category_name ; (2)
value_decl : 'Value' category_name value ; (3)
entity_decl : 'Entity' entity_name 'is' Status (4)
|
| data_decl*
| actions_decl
| behaviour_decl
;

data_decl : 'Data' data_type '=' data_value ; (5)
actions_decl : 'Actions' (send_action | receive_action | leak_action | collect_action | internal_action)* ; (6)
behaviour_decl : 'Behaviour' behaviour+ 'init' behaviour_name ; (7)

```

```

send_action : action_name ':' 'Send' '(' sender ','
| receiver ','
| protocol_name ','
| data_value
| ')' ;

receive_action : action_name ':' 'Receive' '(' sender ','
| receiver ','
| protocol_name
| ')' ;

leak_action : action_name ':' 'Leak' '(' sender ','
| receiver ','
| data_value
| ')' ;

collect_action : action_name ':' 'Collect' '(' sender ','
| receiver ')' ;

internal_action : action_name ':' 'Internal()' ;

```

```

behaviour : behaviour_name '=' behaviour_options ;
behaviour_name : Id ;
behaviour_options : action_name '.' end_seq | action_name '.' behaviour_options | behaviour_options '+' behaviour_options | '('
behaviour_options ')' ;
end_seq : '0' | behaviour_name ;

```

FIGURE 3.12: Grammar Definition: Actions and Behaviour

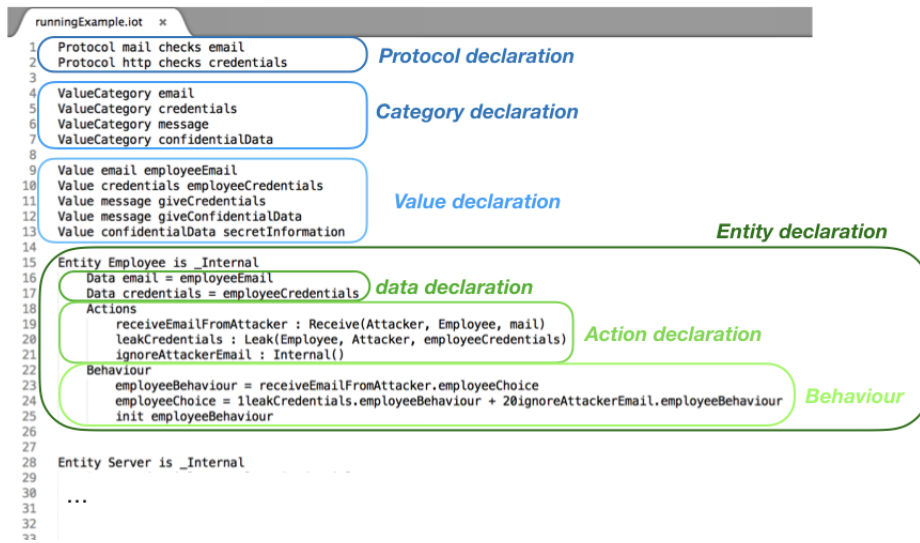


FIGURE 3.13: Grammar Example

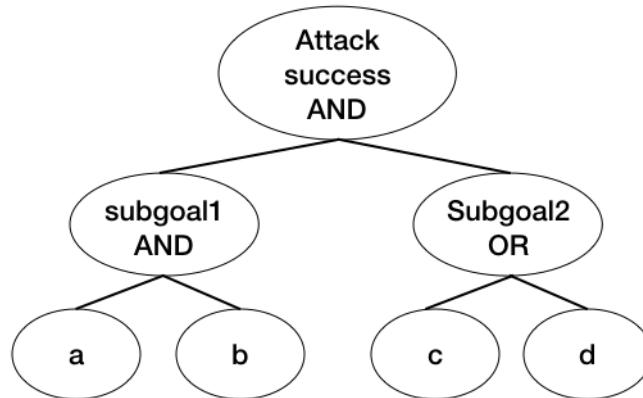


FIGURE 3.14: Attack Tree Example

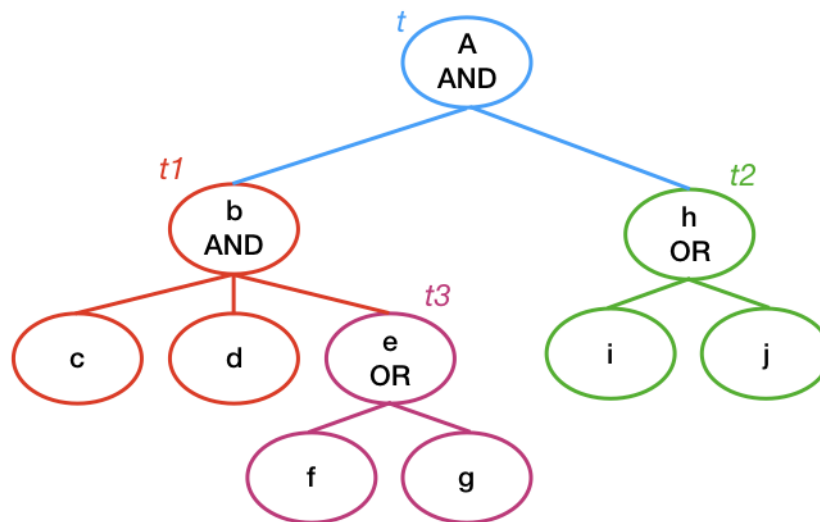


FIGURE 3.15: Attack Tree Example for Height and Depth

Chapter 4

Simulation Approach

In this chapter we focus on the transformation of an IoT model written using SOML into an executable model (a BIP model). This BIP model connects the attack model and the IoT system model.

We start by introducing in Section 4.1 the BIP language, first formally and then using a concrete syntax. We study several aspects of the BIP language such as how BIP components communicate with each other, or how to write a BIP file.

Section 4.2, shows the different elements of the transformation. After an overview of the technical implementation of the parser we see how the transformation is operated, from an IoT system model to a BIP model. This transformation has a formal representation, which we develop in Section 4.3. We also detail the bisimulation between the two languages, first formally, and then using an example for a more concrete explanation.

Finally, Section 4.4 explains how the attack tree is turned into a BIP component and connected to the rest of the model. This will help us understand the role of the attack tree in the modeling process.

4.1 BIP Language

The BIP framework is a component based model introduced in [26]. BIP stands for *Behavior, Interaction and Priority*, that is, the three layers used for the definition of components and their composition in this framework. In this section we describe the key concepts of BIP.

BIP allows the construction of complex, hierarchically structured models from atomic components characterized by their *behaviour* and their *interfaces*. Such components are transition systems enriched with data. Each time a transition is taken, component data, modeled as variables, may be assigned new values computed by user-defined functions (in C). Atomic components are composed by layered application of *interactions* and *priorities*.

Interactions express synchronization constraints and data transfer between the interacting components. Priorities are used to filter among possible interactions and to steer the system evolution so as to meet performance requirements e.g., to express scheduling policies.

The use of the BIP language can be explained by the necessity of running the IoT model we want to perform analysis on. SOML possesses an abstract and a concrete semantic and BIP makes possible for us to run the models with respect to those rules we defined. Indeed, the BIP model that is generated from the SOML model behaves in the same way as we prove it in the Subsection 4.3.3. The choice of BIP resides in the fact that first, the key concepts of BIP makes the transformation towards an equivalent model possible. Also, the BIP Engine and its functionalities allows us to run the simulations as well as to keep traces of them and therefore makes us able to perform the Statistical Model Checking as we see in Chapter 5.

4.1.1 BIP Abstract Syntax

We consider the BIP components to be safe Petri Nets. Therefore, before going into the details of the BIP components, we need to define the Petri Nets.

Petri Nets Introduction

Petri Nets is a modeling language for distributed systems. Petri Nets [57] are represented as state-transition systems. Petri Nets are composed of places, transitions and arcs, as we can see in Figure 4.1. The places represent the states of the system, transitions are actions of the system and both are connected by directed arcs. In Figure 4.1 we identify four places: L1, L2, L3 and L4. To go from place to place we need to pass through transitions. There are three transitions in the example: T1, T2 and T3.

Places contain a discrete number of tokens, that move from place to place with the execution of actions. The tokens condition the transitions and can be used to model concurrent systems. Petri Nets transitions can be labeled with weights. They represent the maximum amount of tokens authorized to go through during a transition. In the same way, places can be labeled with a *capacity* that gives the highest number of tokens that they can hold at once, this is used to model concurrent system. In our case, we considered that both weights and capacities values are 1, as we don't try to model concurrent systems, and therefore we don't need to write them explicitly.

In the general case, a transition is possible when the amount of tokens hold in each of its input places is at least equal to the weight of the arc connecting the place to

the transition. A possible transition may fire at any time. When fired, the tokens in the input places go to output places, depending on the arc weights and output place capacities. Each transition results in a new distribution of the tokens. The token distributions are called *markings*.

The Petri Nets we consider in this subsection are called *safe* Petri Nets and they have the property that no reachable markings puts more than one token in any place.

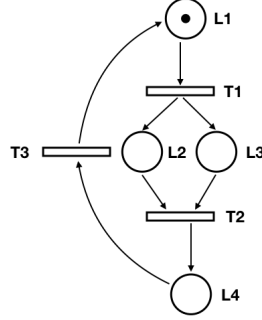


FIGURE 4.1: Petri Net Example

Definition 6 (Petri Net). A Petri Net consists of the tuple $\mathcal{N} = (L, L_0, T, F)$, where

- L is a finite set of places,
- L_0 is the markings,
- T is a finite set of transitions,
- the places L and transitions T are disjoint ($L \cap T = \emptyset$),
- $F \subseteq (L \times T) \cup (T \times L)$ is the flow relation.

The flow illustrates how the system is executed. For instance, going from L_1 to $\{L_2, L_3\}$ is possible in two steps of execution. The first step is the one that enables T_1 , and the second one executes T_1 and arrives to $\{L_2, L_3\}$.

Example 3. Let us transcribe the example of the Figure 4.1.

$$N = (L, L_0, T, F)$$

with

$$L = \{L_1, L_2, L_3, L_4\}$$

$$T = \{T_1, T_2, T_3\}$$

$$F = (\{L_1\}, \{T_1\}), (\{T_1\}, \{L_2, L_3\}), (\{L_2, L_3\}, \{T_2\}), (\{T_2\}, \{L_4\}), (\{L_4\}, \{T_3\}), (\{T_3\}, \{L_1\})$$

$$L_0 = \{L_1\}$$

We define *markings* as the set of functions $m : L \rightarrow \{0, 1\}$. Given two markings m_1, m_2 we define inclusion $m_1 \subseteq m_2 \iff$ for all $l \in L$, $m_1(l) \subseteq m_2(l)$.

Now that the markings are fixed, we can simplify the definition of the Petri Net N . We redefine transitions as a set of input places and output places: T is a finite set of transitions $T = (\bullet t, t\bullet)$, where $\bullet t$ are the input places of T and $t\bullet$ are the output

places of T .

We define the token flow relation by $F = \{(L, T) \mid L \subseteq \bullet t\} \cup \{(T, L) \mid L \subseteq t^\bullet\}$, where:

- L is a set of places
- T is a set of transitions
- $\bullet t$ are the input places of T
- t^\bullet are the output places of T

We consider that F can be deduced from T and therefore we define the extended safe Petri Net by (L, L_0, T) .

Example 4. Using the new definition, let us transcribe the Petri Net of the Figure 4.1.

$$N = (L, L_0, T)$$

with

$$\begin{aligned} L &= \{L_1, L_2, L_3, L_4\} \\ L_0 &= \{L_1\} \\ T &= \{T_1, T_2, T_3\} \\ T_1 &= (\{L_1\}, \{L_2, L_3\}) \\ T_2 &= (\{L_2, L_3\}, \{L_4\}) \\ T_3 &= (\{L_4\}, \{L_1\}) \end{aligned}$$

Definition 7 (Union of Petri Nets). We consider a set of Petri Nets $N_i = (L_i, L_{0,i}, T_i)$, with $i \in I$, where I is a set of indexes such that $I = 1, \dots, n$, with n the number of Petri Nets.

We consider the union of these Petri Nets to be a Petri Net $\mathcal{N} = \cup_{i \in I} N_i$. \mathcal{N} is defined by: $\mathcal{N} = (\mathcal{L}, \mathcal{L}_0, \mathcal{T})$, where:

- $\mathcal{L} = \cup_{i \in I} L_i$
- $\mathcal{L}_0 = \cup_{i \in I} L_{0,i}$
- $\mathcal{T} = \cup_{i \in I} T_i$

Preliminary Definitions

Let us have a set of data domains denoted $\{D_i\}_i$, including several domains, such as \mathbb{N} , etc.. and including the Boolean domain $D_{\text{Bool}} = \{true, false\}$.

Let us suppose that we have a set of variables V .

Let \mathbb{Expr} be a set of operators, that we denote e_1, e_2, e_3, \dots

We denote by $\mathbb{Expr}[V]$ the set of expressions constructed from a set of variables V and operators in \mathbb{Expr} .

We designate by $\mathbb{Asgn}[V]$ a set of assignments to variables in V . An assignment is denoted by $v := e$ and included in the domain $V \times \mathbb{Expr}[V]$. As an expression is a function on a set of variables, we can write $v := f(V)$, whenever $v := e$ with

$e := f(V)$ and $e \in \mathbb{Expr}[V]$.

A *valuation* for the variables in V is a function $\mathbf{X} : V \rightarrow \cup_{j \in J} D_j$ which assigns values to variables. We denote $\mathbf{X}(v)$ the valuation of the variable $v \in V$ and $\mathbf{X}(e)$ the valuation of the expression $e \in \mathbb{Expr}[V]$ according to values of V .

We distinguish between two types of variables: the *deterministic* variables and the *random* variables, used for encoding the stochastic behavior. A deterministic variable is assigned a value through a function. On the other hand, a random variable v is associated with a probability distribution μ over its valuation domain D , denoted as $v \sim \mu$, where $\mu : D \rightarrow [0, 1]$ and $\sum_{x \in D} \mu(x) = 1$.

$|\cdot|$ denotes the cardinality of a set.

Stochastic Atomic Components

Let us give a formal definition of a BIP system, in particular of atomic components and their composition.

Definition 8 (Stochastic Atomic Components). A stochastic atomic component consists of the tuple $\mathcal{B} = (P, V, N)$, where

- P is a set of communication ports. We distinguish respectively input ports $P^{in} \subseteq P$, output ports $P^{out} \subseteq P$ and internal ports $P^{internal} \subseteq P$. We assume they are disjoint, $P^{in} \cap P^{out} \cap P^{internal} = \emptyset$. For every port $p \in P^{in} \cup P^{out}$ we denote by X_p the subset of variables exported and available for interaction through p .
- $V = V^d \uplus V^p$, with
 - $V^d = \{v_1^d, \dots, v_n^d\}$ a set of deterministic variables,
 - $V^p = \{v_1^p, \dots, v_m^p\}$ a set of random variables with an associated probability distribution $v_i^p \sim \mu_i$, for $i \leq m$.
- $N = (L, L_0, T)$ is a Petri-Net where
 - L is a set of places,
 - $L_0 \subseteq L$ is a set of initial places,
 - T is a finite set of transitions $t = (\bullet t, \langle p, g, f \rangle, t^\bullet)$.
 $\bullet t$ (resp. t^\bullet) represent the set of input (resp. output) places of t .
 Transitions are labeled by the triple $\langle p, g, f \rangle$ where:
 $p \in P$ is the port triggered by t , $g \in \mathbb{Expr}[V]$ is the *guard* of t and
 $f = (f^d, R^p)$ is the *update* function of t , such that
 $f^d = \{v := f(V) \mid v \in V^d\} \in \mathbb{Asgn}[V]$ is a set of functions that update the deterministic variables and $R^p \subseteq V^p$ is a subset of random variables to be updated.

A transition is possible only if the guard is *true*. Note that the update for deterministic variables is made via a function whereas the random variables are updated thanks to the random function. We sometimes write p_t, g_t and f_t^d, R_t^p for the label of a transition t .

4.1.2 BIP Semantics

Let us now focus on the execution of a BIP model through the semantic of its atomic components.

We consider a SBIP component $\mathcal{B} = (P, V, N)$. P are the ports of the component, V are its variables and N the Petri Net describing its behaviour.

\mathcal{B} uses a transition system \mathcal{M} obtained from its Petri-Net N . The states in \mathcal{M} are of the form (m, \mathbf{X}) where m is a marking of the Petri-Net N and where \mathbf{X} is a valuation of V . The random variables engender a probabilistic choice over transitions of \mathcal{M} . To illustrate these semantics, we use a simple example of a BIP component described in Figure 4.2, taken from [7].

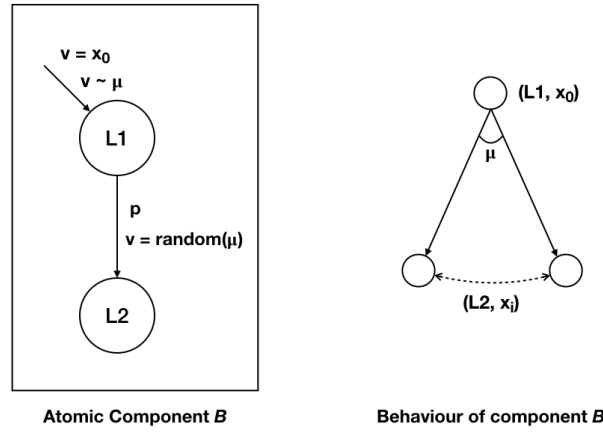


FIGURE 4.2: Example of a stochastic atomic component \mathcal{B} and its behaviour

Figure 4.2 represents a component \mathcal{B} , which Petri Net includes two states: $L1$ and $L2$. \mathcal{B} has a transition going from $L1$ to $L2$ using port p and possesses a probabilistic variable v .

At his initial state, $L1$, the variable v is initialized with the value x_0 . The transition on p updates the random variable v according to a distribution μ . We consider the guard on this transition to be always true, which means that p is always available.

As we can see on Figure 4.2, on the part describing the behaviour of \mathcal{B} , there are several possible transitions from state $(\{l_1\}, x_0)$ to states $(\{l_2\}, x_i)$ for all $x_i \in D$, where D is the valuation domain of v . The probabilities of these transitions is given by μ .

Atomic components with random variables lead to behaviours that combine both stochastic and non-deterministic aspects. A transition is possible if a communication is ready on its associated port. For a given state, several ports can be available and therefore several transitions are possible whether they are associated with random variables or not.

Non-determinism is always resolved in SBIP to a probabilistic choice on an uniform distribution. To formally state this, consider a stochastic component $\mathcal{B} = (P, V, N)$ with $N = (L, L_0, T)$ its Petri-Net and let m be a marking.

Definition 9 (Enabled). We denote with $\text{Enabled}(m; \mathbf{X})$ the set of transitions in T that are enabled by m for a valuation \mathbf{X} :

$$\text{Enabled}(m; \mathbf{X}) = \{t \in T \mid \bullet t \leq m \text{ and } \mathbf{X}(g_t) \text{ is true}\}.$$

Remark that $|\text{Enabled}(m; X)|$ can be greater than one.

In the associated semantics of \mathcal{B} , first a transition is selected with a uniform distribution from the set of enabled transitions. Then, the next state is selected according to the distributions attached to the random variables.

Definition 10 (Semantics of a stochastic atomic component). The semantics of a stochastic atomic component $\mathcal{B} = (P, V, N)$ with $N = (L, L_0, T)$ its Petri-Net and with \mathbf{X}_{init} an initial valuation, is defined as a probabilistic transition system \mathcal{M} , such that:

$\mathcal{M} = \langle Q, P, \pi, q_0 \rangle$, where:

- Q is the set of states, where each state (m, \mathbf{X}) consists of m , a marking, and of \mathbf{X} , a valuation of the variables in V .
- P is the set of ports.
- π is a set of transitions such that $\pi \subseteq Q \times P \times Q$ and defined as follows:
 For $t \in T$, $\bullet t \subseteq m$, $m' = m \setminus \bullet t \cup t^\bullet$, $\mathbf{X}(g_t) = \text{true}$.
 Concerning the variables, we consider the deterministic variable v^d such that $v^d \in V^d$ and the probabilistic variable v^p such that $v^p \in R_t^p, v^p \sim \mu$.
 After the transition, both kind of variables are updated. A deterministic variable uses the update function and a probabilistic variable uses the random function.
 We define the variables update by: $\mathbf{X}' = [v^d := \mathbf{X}(f_t^d), v^p := \text{random}(\mu)]$.
 We express t : $(m, \mathbf{X}) \xrightarrow{pt} (m', \mathbf{X}')$
- $q_0 = (m_0, \mathbf{X}_{\text{init}})$ is the initial state.
 In the Petri Net, m_0 is the marking associated to L_0 , i.e. $m_0(l) = 1 \iff l \in L_0$ and 0 otherwise.

Lastly, we defined the probability of a transition as follows:

$$\mathbb{P}(q \xrightarrow{p} q') = \frac{1}{|\text{Enabled}(m; \mathbf{X})|} \cdot \prod_{v_i \in R^p, v_i \sim \mu_i} \mu_i(X'(v_i)).$$

In the definition above we say that the state (m', \mathbf{X}') is a successor of state (m, \mathbf{X}) , if t is a transition of T enabled by the marking m , the guard g_t evaluates to *true* and the new valuation \mathbf{X}' on the variables $V^d \cup V^p$ is obtained by applying f_t^d on the deterministic variables V^d and updating the random variables in R_t^p . The probability of a transition, as explained above, is given by an uniform distribution on the enabled interactions and by the probabilistic distribution of the random variables updated during the interaction.

4.1.3 Interaction Between BIP Components

Now that we defined the syntax and the semantics of BIP components, let us consider the interactions between these components. Data are transferred between components through their external ports. For simplicity and because it is consistent with our models, we consider the data transfer functions to be deterministic.

Definition 11 (Interaction between components). We consider a set of BIP components $\mathcal{B}_i = (P_i, V_i, N_i)$, with $i \in I$, where I is a set of indexes.

We define an interaction between these components by: $\gamma = (P_\gamma, G_\gamma, F_\gamma)$ where:

- P_γ is a disjoint set of ports: $P_\gamma = \{p_i \mid p_i \in P_i, i \in I\}$.
Each BIP component that takes part in the interaction has exactly one port included on the set.
- G_γ is the global guard of the interaction. The verification is made on variables that belong to the BIP components, thus it is defined on $V_\gamma = \cup_{i \in I} V_i$.
- F_γ is the global update function that defines the exchange of value between the components.
 $F_\gamma = \{v := F(V_\gamma) \mid v \in \cup_{i \in I} V_i^d\}$, with V_i^d a set of deterministic variables such that $V_i^d \subseteq V_i$

Definition 12 (Composition of components). We consider a set of n BIP components $\mathcal{B}_i = (P_i, V_i, N_i)$, with $N_i = (L_i, L_{0,i}, T_i)$ and $i \in I$, where I is a set of indexes, with $I = \{0, \dots, n\}$.

We define Γ as the set of interactions between these components. The resulting of these interactions is a composite component $\Gamma(\mathcal{B}_1, \dots, \mathcal{B}_n)$ where a component $\mathcal{B} = (\Gamma, V, N)$ is defined as follows:

- $\Gamma = \cup_{j \in J} \gamma_j$ is the set of interactions γ_j in which the components takes part, with $j \in J$, where J is a set of indexes, and $J = \{0, \dots, m\}$ with m the highest number of interactions possible.
- $V = \cup_{i \in I} V_i$ is the union of the sets of variables from all the components \mathcal{B}_i
- N is a Petri Net defined as $N = (L, L_0, T)$ with:
 - $L = \cup_{i \leq n} L_i$ is the disjoint set of places of the component
 - $L_0 = \cup_{i \leq n} L_{0,i}$ is the disjoint set of initial places
 - T is the set of transitions defined by $T = \{(\bullet T_\gamma, \langle \gamma, g, f \rangle, T_\gamma^\bullet) \mid \gamma \in \Gamma\}$
with $T_\gamma = \{t_i \mid p_i \in P_\gamma\}$ is the set of transitions that compose the interaction $\gamma \in \Gamma$.
 $\bullet T_\gamma$ and T_γ^\bullet represent respectively the input and output places of T_γ with:
 $\bullet T_\gamma = \cup_{t \in T_\gamma} \bullet t = \{l \mid l \in \bullet t_i, t_i \in T_\gamma\}$ and
 $T_\gamma^\bullet = \cup_{t \in T_\gamma} t^\bullet = \{l \mid l \in t_i^\bullet, t_i \in T_\gamma\}$
 Each transition is labeled using the triple $\langle \gamma, g, f \rangle$ where:
 - * γ is the label of the transition and includes the names of the ports that interact and allow this transition
 - * $g = G_\gamma \wedge (\bigwedge_{t_i \in T_\gamma} g_{t_i})$ is the guard of the interactions that includes the global guard of the transition as well as the local guards of the transitions of each component taking part in the interaction.
 - * $f = (\sqcup_{t_i \in T_\gamma} f_{t_i}) \circ F_\gamma$ is the update function of the interaction, where F_γ is the global update function on the interaction, it can, for instance, represent an exchange of value during the interaction, and $\sqcup_{t_i \in T_\gamma} f_{t_i}$ is the disjoint union of all the local update functions of the components that are part of the interaction. These functions update values within the components during the transition triggered by the interaction, but are not part of the interaction.

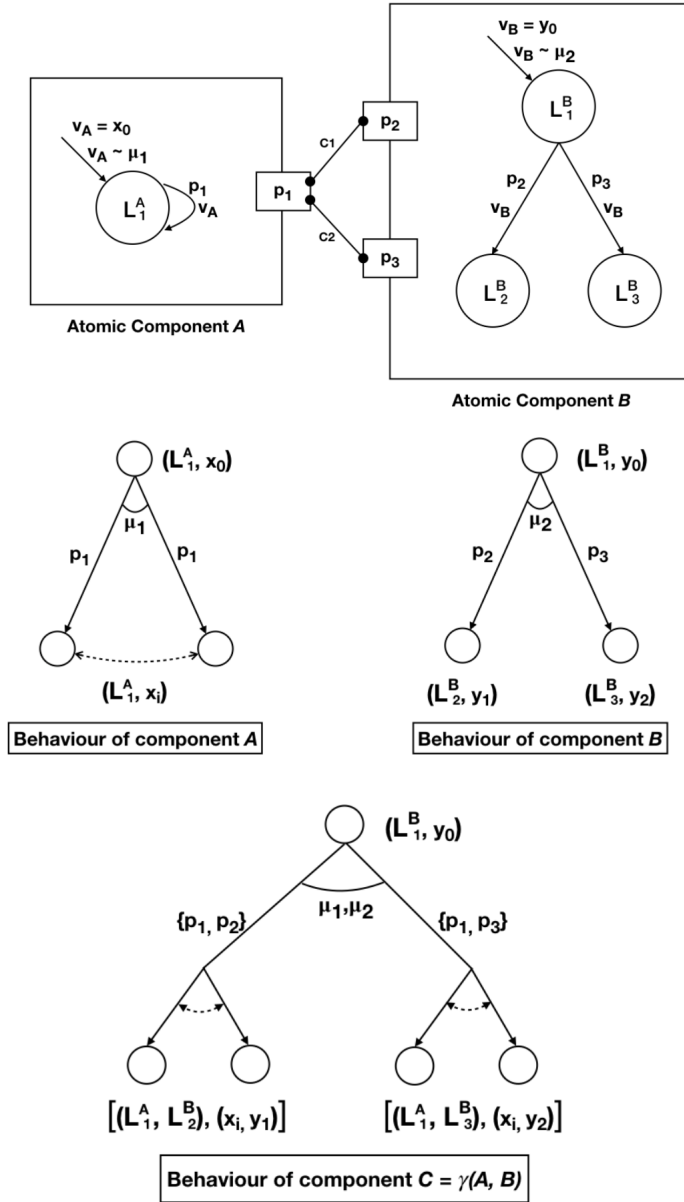


FIGURE 4.3: Example of Component Composition in BIP

Illustrating this definition, Figure 4.3 shows the semantics of a BIP composition. The first part represents the two separate components: A and B connected through their external ports. We recall that in BIP, the transitions are triggered by the ports. Both components have a probabilistic behaviour. For component A, the probabilistic variable v_A conditions the transition on port p_1 . If this transition is fired, then the port of the component A will synchronize with the component B using either the port p_2 , or port p_3 , depending on the value of the probabilistic variable v_B . We see in Figure 4.3, the separate behaviours of the components. On the last part of the figure, we see the behaviour of the component C which is the result of the composition of components A and B. Concerning the resulting behaviour of component C, two combinations are possible: $p_1 - p_2$ or $p_1 - p_3$. The choice between these two is a uniform choice, made by the BIP engine.

4.1.4 BIP Concrete Syntax

In order to better understand the transformations that we detail in the two following sections, it seems important to know what are the concepts expressed in BIP files. To do so, we use a simple example that represents two components, denoted *A* and *B*. Figure 4.4 shows the BIP code of the example and Figure 4.5 shows its graphical representation.

The BIP compiler makes it possible to use C library within the BIP code. This means that we can use existing functions and data types, or define them in an external C file and they are available for us to call from the BIP file.

```

(1) @cpp(include="stdio.h",src="ex.c")
    package AB_connection
(2) extern data type ..
    extern function ..
(3) port type p_data(int d)
    Port type p()
    connector type p_connector (p s, p r)
    define s r
    up {}
    down { r.d = s.d; }
    end
(4) atom type A(int data)
    data d
    export port p_data p1(d)
    port p p2()
    place A0, A1
    initial to A0 do{ d = data; }
    on p1 from A0 to A1
    on p2 from A1 to A0
    end
(5) atom type B()
    data d
    export port p_data p1(d)
    place B0, B1
    initial to B0
    on p1 from B0 to B1
    provided (d > 1)
    do { d = d+1; }
    end
(6) compound type AB_connection()
    component A a(3)
    component B b()
    connector p_connector p_c(A.p1, B.p1)
    end

```

FIGURE 4.4: BIP simple example code

In the BIP file of the Figure 4.4, this is expressed by (1) the import of the C libraries and external C file and by (2), the declaration of the external functions that are used within the BIP file. The rest of the file is used for defining the BIP components and their interactions.

In (3) we see the ports and connectors definition. We recall from the previous subsection that BIP components communicate with each other through ports. BIP offers the possibility to define *types* of ports. In Figure 4.4 we define two different types of port. The type *p_data* that has one integer argument and the type *p* no arguments. As it does for the ports, BIP allows the definition of several kind of connectors. The

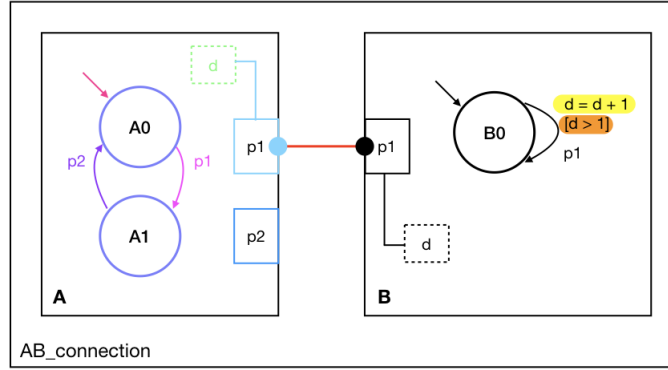


FIGURE 4.5: BIP simple example

connectors take as parameters all the types of ports they link and define the actions performed when the connection happens. So, we need to define a connector that allows components to communicate by linking the types of ports we defined and describing what occurs during the connection. In our example, the components communicate using only the *p_data* ports so the *A* component can send the value *d* to the component *B*. We only need to declare one connector type: *p_connector*. The *p_connector* type is a connector with two arguments which are two *p_data* ports. This indicates that this connector type can link two *p_data* ports. It is possible to declare a connector type that can link more than two ports of different types. Now that we defined the correct arguments, which specifies the number and types of ports that are linked, we need to describe the actions that happens when the ports are connected. We do this in the *up* and *down* parts of the connector.

First, let's consider the *down* code block. It is used to describe an exchange of information when a *sender* component wants to broadcast content to one or several other components that are *receiver* components. In our example, the sender port *s* transfers a value to the receiving port *r*. The *up* code block, provides a way to use the connector as a port

Part (4) shows the declaration of a BIP component. When instantiating an component of type *A*, we have to give it a parameter value in order to initialize the variable *d*. The component *A* declares two ports. The first one is a *p_data* port preceded by the keyword *exported*. This indicates that this port is able to communicate with the outside world. This port also has a parameter which is the component variable *d*. Therefore, any communication through this port involves the variable *d*. Depending on *A*'s role in the communication, either the value of variable *d* is sent, either *d* is updated with a received value. The second port is not exported, we call it an *internal* port. It is not used in order to communicate with other components but rather to execute an internal action.

The places represent the possible states of the component. The initial and final state is *A0* and there is an extra place: *A1*.

In the definition of the *B* type component in (5), we see several new concepts appear. The yellow part is a *guard*: it is a condition for the transition to take place. Here, the condition is on the value of the variable *d* that must be superior to 1. The orange part is an *update function* which updates the value of the variable *d* at the moment of the transition.

Finally, the part (6) shows the compound creation. Earlier, we created types of components and ways of connecting them via the port and connector types definitions. In the compound, we instantiate objects and execute the model. So we create two

atomic components, one of type A and one of type B and compose them. For the A component, we initialize the variable $d = 3$. Then we create an instance of the *p_connector*, called *p_c*, to which we give as parameters the names of the exported ports of the components. This instance of the connector triggers the communications between the two BIP components.

4.2 IoT Model to BIP

IoT Security Oriented Description Language is meant to model an IoT system from a security point of view. It has a syntax of its own, as well as an operational semantics. BIP allows us to run simulations of the system behaviour with respect to these semantics rules. To be able to execute an IoT model, we decided to use the BIP engine. To do so, we need to have a corresponding BIP model to run. Previously, we detailed the technical functioning of the parser. In this section, we first see how we implemented a parser, based on the concrete syntax of IoT SOML, that converts an IoT model to a BIP model. We go into details on the parsing process by explaining the creation of the *lexer* and the *parser* and their roles in Subsection 4.2.1.

We then delve into the details of the transformation itself, in Subsection 4.2.2 by describing the BIP code the parser generates, depending on the IoT concepts expressed in IoT models. We illustrate the transformation using the running example in Subsection 4.2.3.

4.2.1 Parser Implementation

After defining the concrete syntax, in Subsection 3.3.4, we need to implement a tool that can recognize a file written using this grammar and transform it into a BIP File. This tool, called a *parser* or a *syntax analyzer*, is able to identify the different elements of the SOML file, for instance the entities or the protocol definitions. From there it generates BIP elements as we see in more details in Subsection 4.2.2.

To create the foundations of the parser, we decided to use ANTLR4 [56]. ANTLR4 is a parser generator created by Terence Parr. There are several reasons for why we decided to use a parser generator, and ANTLR4 in particular. First IoT Modeling Language is evolving, and being able to generate parts of the parser automatically for each change saved us a considerable amount of time. Using ANTLR4 also helped us minimize the potential errors, which was important considering that these files are the foundation of the parser. Finally, the code generated by ANTLR4 is clear and gives us total control over multiple aspects of the parsing process such as error reporting, error recovery, etc. ANTLR4 equally provides several output languages, we chose to implement the parser in Java.

Therefore we used ANTLR4 to generate a *parse tree* and its associated *tree walker*. The parse tree, also called *Abstract Syntax Tree*, is in charge of recognizing the grammar patterns. The tree walker provides ways to exploit these patterns, for instance by generating the corresponding BIP code. But to better understand what they represent, let's take a closer look to ANTLR4's process and what parsing really is.

The parsing process results in the generation of a parse tree which requires two steps as we can see in Figure 4.6. From the g4 file in which we specified the grammar, ANTLR4 first creates a *lexer* and then a parser that are specific to the IoT Modeling Language. The lexer, or *Lexical Analyzer* is in charge of transforming a sequence of characters into a sequence of *tokens*. A token is a string that corresponds to a lexical symbol, which means that it is a character sequence with an identified meaning. It

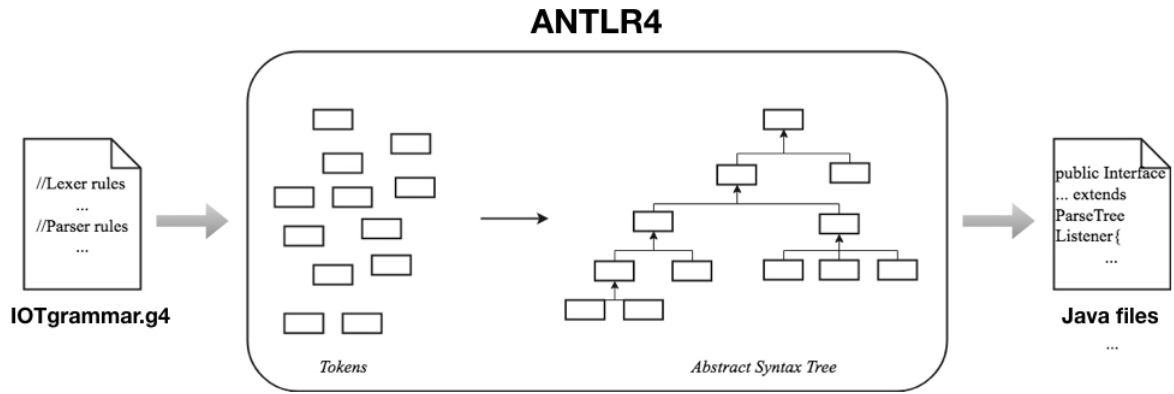


FIGURE 4.6: ANTLR4

consists of a pair $\{name; value\}$. The name is the category of the lexical unit and the value may or may not be null.

Then, ANTLR4 generates the parser, or syntax analyzer. The parser's role concerns the second part of the analysis: from the tokens previously formed, the Parser builds the parse tree. The parse tree records how the parser recognized the structure of the input (the IoT file) sentences and component phrases. We can use our running

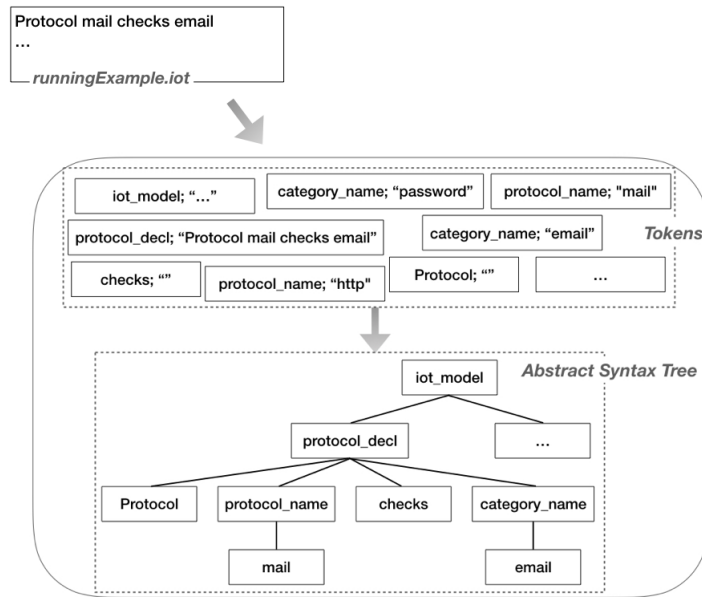


FIGURE 4.7: Lexing and Parsing

example to illustrate the above explanation, as shown in Figure 4.7. When describing the system with the IoT Modeling Language we start by the global declaration of the protocols. The running example starts with the declaration of the mail protocol this way: "Protocol mail checks email". The terms "Protocol" and "checks" are fixed, unlike "mail" and "email" which are the name of the protocol and the category of data on which we perform the verification. The Lexer recognises the protocol declaration pattern and can associate it to its value. The same thing happens for the "protocol_name" and the "category_name". However, "Protocol" and "checks" being keywords of the protocol declaration don't have an associated value.

So now we are able to parse an IoT file and generate a parse tree. We then need to exploit this tree in order to generate a BIP model that we can execute. ANTLR provides us a way to do so with the combined use of the tree walker and the *parse tree listener interface*. In Java, event listeners are interfaces responsible to handle events. This interface provides enter and exit methods for each rule we defined in the grammar. It then responds to events triggered by the built-in tree walker. This means that when the tree walker goes through the syntax tree, it triggers two events. One event is triggered when the walker first meets an object, before exploring it. The second event is triggered when the walker leaves the object after going through it. To this extend, we can walk through the tree and associate actions, for example the creation of BIP elements, every time we trigger events in the listener implementation. Figure 4.8 shows in more details the java files generated by ANTLR4.

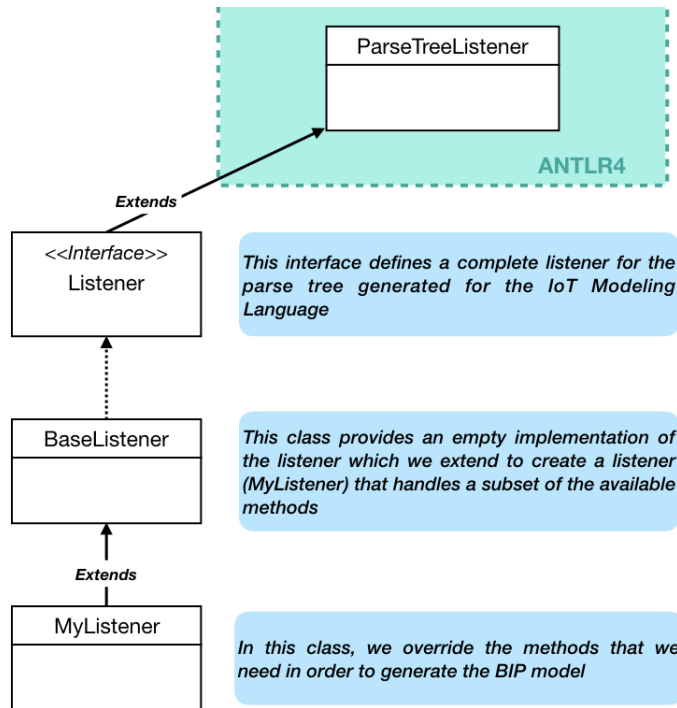


FIGURE 4.8: Java Classes generated by ANTLR4

4.2.2 Model transformation

The IoT model specifies a certain amount of information about the system. This information must be relevant and adequate for the parser to be able to build a well formed BIP file that behaves during its execution in the way that we specified in the IoT model. Figure 4.9 shows how IoT objects and BIP objects are linked. We recall that in BIP, it is mandatory to create a type for objects (ports, connectors, components, ...) in order to instantiate them. All the information concerning the types of BIP objects can be found in the IoT file.

As a preliminary, it is necessary to indicate that BIP language handles certain data formats such as string, int and float. In order to use another format, the user has to define it by himself, in C and import the C file in the BIP model as an external library. For our model we created a `string_set` type, which is a one dimension string

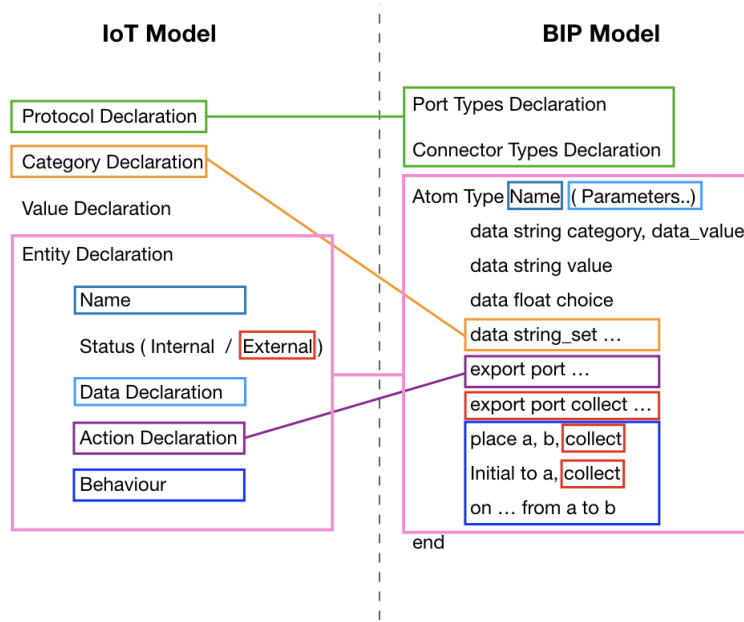


FIGURE 4.9: IoT Equivalence in BIP

table, and defined several functions to manipulate it.

For reason of clarity we start with the entity declaration of the IoT files. An entity declaration results, in the BIP model, in the creation of a type of atomic component. The name of the entity becomes the name of the component type.

In IoT, we declare the initial knowledge of an entity as a series of data declaration. In BIP, the knowledge is represented as a series of `string_set`. For each existing category of data declared in the IoT file, the BIP components have corresponding `string_set`. The initial knowledge is given as parameters to the components and added, at the initial state, to the `string_set` of their category. This choice of design satisfies the need of having data category in IoT, which is not possible in BIP. Indeed in IoT Modeling Language one can create all the data categories needed. During the exchanges of information the entities are able to send and receive all the data, regardless of their category. This is why each component has a data set for each category: it has to be able to receive every kind of data and add it to its knowledge. This also facilitates protocol verifications. As the protocol verification is performed on a certain category of data. In BIP, we make the verification on the corresponding set of strings: if the sender and the receiver have a common value in their respective sets, then the communication is allowed.

Let us now look at the actions and the behaviours of the entities. In BIP, the actions of the components are specified as a set of transitions. These transitions are triggered by their ports. To enable the interactions between the ports, there are connectors. Both port types and connectors types must be declared prior to their use. So for each protocol declared in the IoT file, there is a port type in the BIP file. In addition, a specific connector type that connects two ports of the same kind is created, allowing the ports to exchange information. Having a specific port for each protocol simplifies the equivalence with the IoT Modeling Language, as we will see in Subsection 4.3.3. The connector also includes the verification on the corresponding type of data. We had no choice but to have the protocol verification in the connector rather than having it on the entity port, using a guard. Indeed, the connector has access to the data of both the receiving component and the sending component and can perform the verification. Whereas a component has only access to its own data and is then not

able to know if the preconditions for a communication are fulfilled. On the component level, for each action of the IoT entity, the BIP component possesses a corresponding port, which type depends on the type of communication. Indeed, if the port is used for a Send/Receive type on, for instance, the http protocol, the BIP component will have a port of type "http_protocol".

Let us now consider the Leak/Collect types of exchange. We first create two types of ports: a leak type and a collect type. A connector type is defined which allows the leak port to transfer data to the collect port, without performing any type of verification. However, the way we attribute these ports to the entities differs. We recall that the entities have a status: they can be either Internal or External. The External entities try to make the Internal entities leak confidential information. So in BIP, we assign a leak port to the internal entities for each leak they perform. The collect ports are automatically given to the external entities, whether they are expressed in the IoT file or not. They have a collect port for each internal entity of the system. This way, when an internal entity leaks an information, the collect counterpart of the communication is performed by its associated port on the Attacker side.

The behaviour is also expressed differently, depending on the status of the entity. Indeed, if the entity is External, the collect is always available. This means that, even in the middle of a sequence of actions, the external entity is able to collect information.

4.2.3 Example of Transformation

Going back to the running example, we can see in Figure 4.10 an internal entity: the Employee and in Figure 4.11 an external entity: the Attacker once they have been turned into BIP components.

Data We can see that both components have a `string_set` of data for each category of the system: email, credentials, message, confidential data. In addition they both have two variables: "category" and "value". The variable "category" is used to check the category of a received value in order to put it in the correct set. The variable "value" is initialized with either the value the component is about to send, or the value it just received.

Ports

- **External Ports:** For each transition performed (outside of the Leak/Collect ones), both components have an external port that corresponds to a transition in their behaviour.
 - **Leak Port:** For the employee, we can see that the BIP component has a leak type of port that corresponds to him, leaking his credentials to the attacker.
 - **Collect Ports:** We can see in Figure 4.11 that the attacker has a "collect" port for each internal entity of the system: one for the employee and one for the server. We can also notice that the "collect" transitions are performed in parallel with their nominal behaviour. This shows that at anytime, the attacker is able to collect information from any internal entity.
- **Internal Ports:** They are used to provide the component a way of making an internal transition, without communicating with another component.

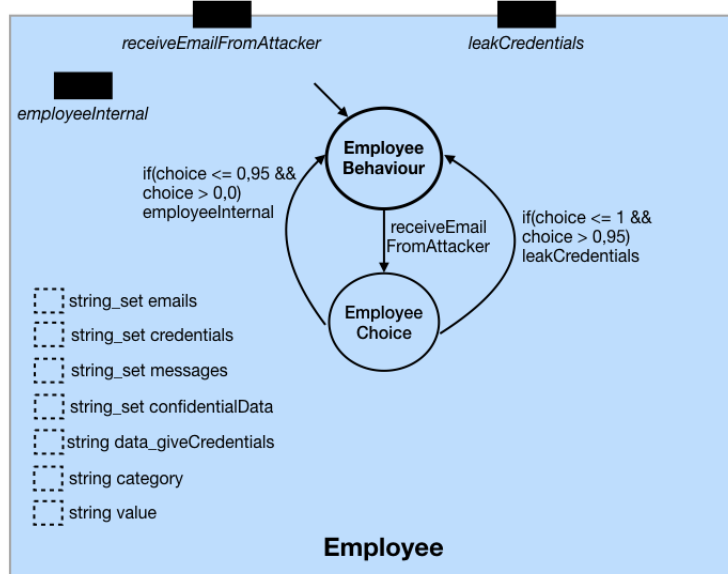


FIGURE 4.10: BIP component of the Employee

Let's now take a look at how the weights we added on the actions of the IoT entities become probabilities on the BIP component transitions.

As we can see in Figure 4.10, from the *EmployeeChoice* state, two transitions are possible: *employeeInternal* or *leakCredentials*. The transition taken depends on the value of variable *choice*. If the value of *choice* is in the $[0;0,95]$ interval, the employee won't leak his credentials, but if *choice*'s value is within $]0,95;1]$ he will then leak them to the attacker.

The value of *choice* is obtained via a random function.

The calculation of the value intervals is based on the weights we assigned to the actions in the IoT file. To transform the weight into probabilities we carry out the following steps:

1. from the current state we check all the possible transitions
2. we calculate the total weight by adding together all the weights of possible transitions
3. each weight is divided by the total weight, which gives a value within $[0;1]$
4. we calculate the intervals using the new weight value

Now that we have a better idea of how the transformation occur thanks to a concrete example, let's see how we define it formally.

4.3 Formal Transformation

During the transformation, entities of an IoT model become atomic components in \mathcal{SBIP} and the communication between them are represented with interactions. To model this we define an atomic component as the union of several Petri Nets, which all have a common set of variables, guards and update functions. The deterministic variables are used to model the entity's knowledge. The random variables, similarly to the transformation of DTMC into \mathcal{SBIP} of [7], encode the probabilities associated to actions in a summation process.

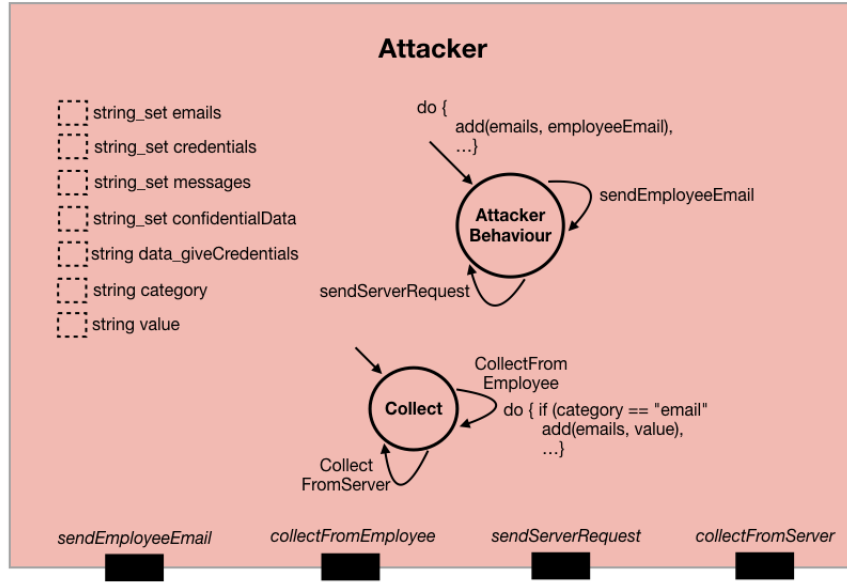


FIGURE 4.11: BIP component of the Attacker

We need to formally prove the *bisimulation* between the IoT model and the \mathcal{SBIP} model. In order to show that this equivalence exists between both languages, we need to keep track of the transformation. The bisimulation shows that for a given transition in the IoT model, a similar transition exists in the \mathcal{SBIP} model.

Consequently, in the transformation we annotate places and random variables using the function ℓ . This function exploits the notations of the original IoT system. In addition, we identify places that have congruent notations, i.e. $l_1 \equiv_L l_2 \iff \ell(l_1) \equiv_P \ell(l_2)$, with l_1 and l_2 being places of the BIP model. We write l_T when $\ell(l) = T$, and v_T when $\ell(v) = T$, with l a place and v a variable of the BIP model.

First we introduce a number of transformation functions used in the formal definition as preliminaries in the Subsection 4.3.1. In Subsection 4.3.2, we propose a definition of the transformation of the behaviour part of the IoT entity. Finally, in Subsection 4.3.3, we formally describe the bisimulation between the two languages.

4.3.1 Transformation Functions

Considering a thread T , we define three functions:

- $\llbracket T \rrbracket_v$ defines **random variables** used by the thread T .
- $\llbracket T \rrbracket_s$ defines **places** for the thread T .
- $\llbracket T \rrbracket_t$ defines the **transitions** for T .

Following each function definition we give an application of it using the attacker's behaviour of the running example that we defined in Example 2.

We will then consider the following thread:

$$\begin{aligned}
 A = & [n_3] \text{attacker} \xrightarrow[\text{giveCredentials}]{\text{mail}} \text{employee}.A + [n_4] \text{attacker} \xrightarrow[\text{giveSecretInformation}]{\text{http}} \text{server}.A \\
 & \underbrace{\hspace{10em}}_{a_1} \qquad \underbrace{\hspace{10em}}_{a_2} \\
 & + [n_5] \underbrace{\text{attacker} \leftarrow \text{employee}.A}_{a_3} + [n_6] \underbrace{\text{attacker} \leftarrow \text{server}.A}_{a_4}
 \end{aligned}$$

These functions can be applied on all types of threads whether they are recursive or not. However on the case of a recursive thread, it must be written on its entirety without deploying the thread in infinity. We have to be able to see all the possible actions of the thread appear for the function application to be correct.

Definition 13 (Random variable Function for Thread). This function takes as input a thread and generates a set of random variables or an empty set.

$$\begin{aligned}
 \llbracket \sum_{i \in I} [n_i] a_i.T_i \rrbracket_v &= \bigcup_{i \in I} \llbracket a_i.T_i \rrbracket_v \cup \{v_T \mid v_T \sim \mu \text{ s.t. } \mu(a_i.T_i) = n_i, \forall i \in I\} \\
 &\text{where } T = \sum_{i \in I} [n_i] a_i.T_i \text{ and } |I| > 1 \\
 \llbracket a.T \rrbracket_v &= \llbracket T \rrbracket_v \\
 \llbracket A \rrbracket_v &= \llbracket 0 \rrbracket_v = \emptyset
 \end{aligned}$$

Whenever a thread T is of the form $\sum_{i \in I} [n_i] a_i.T_i$, we introduce a new random variable v_T , that will allow the system to make a probabilistic choice when necessary between all available options.

The valuation domain D for v_T is the set of states associated to the possible continuations i.e. $D = \{a_i.T_i\}_{i \in I}$. The probability distribution of v_T is defined by the probabilities n_i i.e. $\mu(a_i.T_i) = n_i$.

If T is not a sum, no random variable is necessary as the aim of the random variable is to make a choice between the different threads that compose the sum.

Example 5.

$$\begin{aligned}
 \llbracket A \rrbracket_v &= \llbracket a_1.A \rrbracket_v \cup \llbracket a_2.A \rrbracket_v \cup \llbracket a_3.A \rrbracket_v \cup \llbracket a_4.A \rrbracket_v \cup \\
 &\{v_A \mid v_A \sim \mu \text{ s.t. } \mu(a_1.A) = n_3, \mu(a_2.A) = n_4, \mu(a_3.A) = n_5, \mu(a_4.A) = n_6\}
 \end{aligned}$$

with

$$\begin{aligned}
 \llbracket a_1.A \rrbracket_v &= \llbracket A \rrbracket_v = \emptyset \\
 \llbracket a_2.A \rrbracket_v &= \llbracket A \rrbracket_v = \emptyset \\
 \llbracket a_3.A \rrbracket_v &= \llbracket A \rrbracket_v = \emptyset \\
 \llbracket a_4.A \rrbracket_v &= \llbracket A \rrbracket_v = \emptyset
 \end{aligned}$$

This thread is composed of a choice between four possible actions. So we only need to create one probabilistic variable v_A . The choice between these four actions will be made depending on the value of v_A .

Definition 14 (Place Function for Thread). This function takes a thread as input and generates a set of places.

$$\begin{aligned} \llbracket \sum_{i \in I} [n_i] a_i.T_i \rrbracket_s &= \bigcup_{i \in I} \llbracket T_i \rrbracket_s \cup \{l_T, l_T^*\}, \text{ where } T = \sum_{i \in I} [n_i] a_i.T_i \text{ and } |I| > 1 \\ \llbracket a.T \rrbracket_s &= \llbracket T \rrbracket_s \cup \{l_{a.T}\} \\ \llbracket A \rrbracket_s &= \{l_A\} \\ \llbracket 0 \rrbracket_s &= \{l_0\} \end{aligned}$$

Each thread, regardless of its form, possesses an initial place. The initial place its definition and represents the entry point of the thread.

For a thread T of the form $\sum_{i \in I} [n_i] a_i.T_i$, we introduce two variables l_T and l_T^* where l_T represents the initial state of the thread T and l_T^* denotes the state in which the random variable v_T is updated.

If T is not a sum, we don't need to create an additionnal state for a random variable to be updated. So we only create a place for each possible continuation of the thread.

Example 6.

$$\begin{aligned} \llbracket A \rrbracket_s &= \llbracket a_1.A + a_2.A + a_3.A + a_4.A \rrbracket_s \\ &= \llbracket A \rrbracket_s \cup \{l_A, l_A^*\} \\ &= \{l_A\} \cup \{l_A, l_A^*\} \\ &= \{l_A, l_A^*\} \end{aligned}$$

In this example, we need two places: l_A and l_A^* . l_A represents the initial and the final place of the Petri net.

Definition 15 (Transition Function for Thread). This function takes as input a thread and generates a set of transitions.

$$\begin{aligned} \llbracket \sum_{i \in I} [n_i] a_i.T_i \rrbracket_t &= \bigcup_{i \in I} \left((\{l_T^*\}, \langle a_i, g = (v_T == a_i.T_i), f \rangle, \{l_{T_i}\}) \cup \llbracket T_i \rrbracket_t \right) \\ &\quad \cup (\{l_T\}, \langle \tau, \text{true}, f^* \rangle, \{l_T^*\}) \text{ where } T = \sum_{i \in I} [n_i] a_i.T_i \text{ and } |I| > 1 \\ \llbracket a.T \rrbracket_t &= (\{l_{a.T}\}, \langle a, \text{true}, f \rangle, \{l_T\}) \cup \llbracket T \rrbracket_t \\ \llbracket A \rrbracket_t &= \llbracket 0 \rrbracket_t = \emptyset \end{aligned}$$

where f is the update function such that $f = (f^d, R^p)$, with $f^d = \{v := v | v \in V^d\} \in \mathcal{Asgn}[V]$, and $R^p = \emptyset$ and f^* defined as f but with $R^p = \{v_T\}$. g is the guard on the transition. The guards are only used when making a probabilistic choice: Suppose we are currently running thread T and we wish to go from state l_T^* to a state l_{T_i} . The guard then checks that the value of the random variable v_T is updated to $a_i.T_i$. For the rest of transitions, the guard is the constant *true*. If the thread is not constituted of a sum, the guard is the constant *true* and we only need one transition by action that is part of the thread.

Example 7.

$$\begin{aligned}
\llbracket A \rrbracket_t = & ((\{l_A^*\}, \langle a_1, g = (v_A == a_1.A), f \rangle, \{l_A\}) \cup \llbracket A \rrbracket_t) \cup \\
& ((\{l_A^*\}, \langle a_2, g = (v_A == a_2.A), f \rangle, \{l_A\}) \cup \llbracket A \rrbracket_t) \cup \\
& ((\{l_A^*\}, \langle a_3, g = (v_A == a_3.A), f \rangle, \{l_A\}) \cup \llbracket A \rrbracket_t) \cup \\
& ((\{l_A^*\}, \langle a_4, g = (v_A == a_4.A), f \rangle, \{l_A\}) \cup \llbracket A \rrbracket_t) \cup \\
& (\{l_A\}, \langle \tau, \text{true}, f^* \rangle, \{l_A^*\})
\end{aligned}$$

with

$$\begin{aligned}
f^* &= (f^{*d}, R^p) \text{ where } f^{*d} = \{v := v \mid v \in V^d\} \text{ and } R^p = \{v_A\} \\
f &= (f^d, R^p) \text{ where } f^d = \{v := v \mid v \in V^d\} \text{ and } R^p = \emptyset \\
V^d &= \{v_{email}, v_{credentials}, v_{http}, v_{messages}, v_{confidentialData}\} \\
\llbracket A \rrbracket_t &= \llbracket 0 \rrbracket_t = \emptyset
\end{aligned}$$

In this example, we have one transition for each action. We also have a transition labeled τ that allows us to update the value of the random variable.

Using these three functions we generate a certain number of transitions, places and probabilistic variables that we represent as a Petri Net in Figure 4.12. For simplicity reasons, the transitions of the figure are only partially labeled with the ports and the update functions.

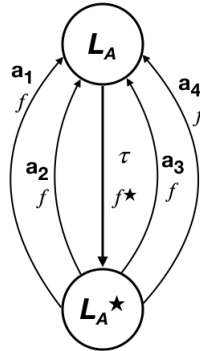


FIGURE 4.12: Petri Net corresponding to the thread A

4.3.2 Entity Formal Transformation

For clarity reasons we specify the transformation of the entity's thread and the entity itself in two different definitions.

First we encode each thread of the entity's process in Definition 16. We use a distinct Petri Net for each thread in order to clearly separate the behaviour of the different threads. We use disjoint union for combining the Petri Nets of the different threads. This is coherent in the BIP model because different threads in an entity cannot interact with each other, but only with other entities. Following, Definition 17 formally describes the composition of a BIP component.

Let us consider an IoT entity described in IoT SOML has having:

- a set of data, possibly belonging to different categories, defining its knowledge
- a set of actions that it can perform

- a behaviour, composed of one or several threads that uses the actions previously defined

Definition 16 (Transformation of a Thread into a Petri Net). To each thread in an entity, we associate a corresponding Petri Net that specify the behaviour of the BIP atomic component. As the behaviour of an entity can be composed of several threads, we can define several Petri Nets. However, if several Petri Nets belong to the same component, they don't share any probabilistic variables or places between them. The probabilistic variables are defined by V^p being the disjoint set of probabilistic variables for the thread T , defined by $V^p = \llbracket T \rrbracket_v$. We define a thread T by the Petri Net $N = (L, L_0, T_N)$ where:

- L is the set of places of T defined by $L = \llbracket T \rrbracket_s$
- L_0 is the set initial places, which represents the entry point of the thread and is defined by $L_0 \subseteq L$. For each separate thread of the behaviour of an entity we create a distinct initial place.
More formally, for a thread T , we define an initial place L_T .
- T_N is a finite set of transitions defined by $T_N = \llbracket T \rrbracket_t$

Now that we defined how each thread is encoded in a Petri Net, let us define the entire BIP component built from the IoT entity. The sets of places, transitions and probabilistic variables that are composing the process of the entities are the disjoint unions of the sets that we defined for the threads.

Definition 17 (Transformation of an IoT Entity into an Atomic BIP component). For an IoT entity E , we define the transformation to be a tuple $B = (P, V, \mathcal{N})$, with

- P are the ports of the atomic component that correspond to the actions of the IoT entity such that $P = P^{in} \uplus P^{out} \uplus P^{internal}$ with P^{in} being the input ports, P^{out} the output ports and $P^{internal}$ the internal ports;
- V are the variables of the BIP component such that $V = V^d \uplus V^p$ where
 - V^d is the set of deterministic variables, $V^d = \{v_c = k_i^c \mid c \text{ is a protocol used in the entity and } k \text{ the initial knowledge of the entity}\}$;
 - V^p is the set of probabilistic variables, such that $V^p = \bigcup_{i \in I} V_i^p$, and $I = \{0, \dots, n\}$ with n the number of threads;
- \mathcal{N} is the union of Petri Nets that constitute the behaviour of the BIP component such that $\mathcal{N} = \bigcup_{i \in I} N_i$.

Example 8. To better understand the formal transformation of an entity let us use as an example the transformation of the employee entity into a BIP component.

The behaviour of the Employee is defined by the recursive threads:

$$\begin{aligned}
k_{employee} &= \{ \text{mail} = \{\text{employeeEmail}\}, \\
&\quad \text{message} = \emptyset, \\
&\quad \text{credentials} = \{\text{employeeCredentials}\}, \\
&\quad \text{confidentialData} = \emptyset, \\
&\quad \text{http} = \emptyset \} \\
E &= \underbrace{\text{employee} \xleftarrow{\text{mail}} \text{attacker} . C}_{e_1} \\
C &= [n_1] \tau . E + [n_2] \underbrace{\text{employee} \xrightarrow{\text{employeeCredentials}} \text{attacker} . E}_{e_2}
\end{aligned}$$

Transformation of the Behaviour of the Employee into a Petri Net

In this example, the thread is recursive. As we explained before, to correctly apply the functions on a recursive thread, it is necessary to unfold the thread. This way, we can see the whole behaviour appear, without deploying it to the infinity. To show the unfolding of the thread, and for readability reasons, we added indices to the thread definition. This way, $\llbracket E \rrbracket^1$ indicates the first time that we see the thread and that it has not been fully deployed. As for $\llbracket E \rrbracket^2$, it shows that the thread E has already been deployed and that it is not necessary to unfold it anymore. Whenever we declare a thread without any indice, it means that the thread has no definition and doesn't need to be deployed, therefore we consider the indice to be 1 and do not write it.

Generation of the random variables:

$$\begin{aligned}
\llbracket E \rrbracket_v^1 &= \llbracket e_1 . C \rrbracket_v \\
&= \llbracket C \rrbracket_v \\
\llbracket C \rrbracket_v &= \llbracket [n_1] \tau . E + [n_2] e_2 . E \rrbracket_v \\
&= \llbracket \tau . E \rrbracket_v \cup \llbracket e_2 . E \rrbracket_v \cup \{v_C \mid v_C \sim \mu \text{ s.t. } \mu(\tau . E) = n_1, \mu(e_2 . E) = n_2\} \\
&= \llbracket E \rrbracket_v^2 \cup \llbracket E \rrbracket_v^2 \cup \{v_C \mid v_C \sim \mu \text{ s.t. } \mu(\tau . E) = n_1, \mu(e_2 . E) = n_2\}
\end{aligned}$$

with

$$\llbracket E \rrbracket_v^2 = \emptyset$$

Generation of the places:

$$\begin{aligned}
\llbracket E \rrbracket_s^1 &= \llbracket e_1 . C \rrbracket_s \\
&= \llbracket C \rrbracket_s \cup \{l_{e_1 . C}\} \\
\llbracket C \rrbracket_s &= \llbracket [n_1] \tau . E + [n_2] e_2 . E \rrbracket_s \\
&= \llbracket E \rrbracket_s^2 \cup \llbracket E \rrbracket_s^2 \cup \{l_C, l_C^*\} \\
&= \{l_C, l_C^*\}
\end{aligned}$$

with

$$\llbracket E \rrbracket_s^2 = \emptyset$$

Generation of the transitions:

$$\begin{aligned} \llbracket E \rrbracket_t^1 &= \llbracket e_1.C \rrbracket_t \\ &= (\{l_{e_1.C}\}, \langle e_1, true, f \rangle, \{l_C\}) \cup \llbracket C \rrbracket_t \\ \llbracket C \rrbracket_t &= \llbracket [n_1]\tau.E + [n_2]e_2.E \rrbracket_t \\ &= (\{l_C^*\}, \langle \tau, g = (v_C == \tau.E), f \rangle, \{l_E\}) \cup \llbracket E \rrbracket_t^2 \cup \\ &\quad (\{l_C^*\}, \langle e_2, g = (v_C == e_2.E), f \rangle, \{l_E\}) \cup \llbracket E \rrbracket_t^2 \cup \\ &\quad (\{l_C\}, \langle \tau, true, f^* \rangle, \{l_C^*\}) \end{aligned}$$

In the transitions generated, we can see that there is a place l_E . Yet, we have $e_1.C \equiv_P E$, and so we consider $l_{e_1.C} \equiv_L l_E$.

$$\begin{aligned} f^* &= (f^{*d}, R^p) \text{ where } f^{*d} = \{v := v \mid v \in V^d\} \text{ and } R^p = \{v_C\} \\ f &= (f^d, R^p) \text{ where } f^d = \{v := v \mid v \in V^d\} \text{ and } R^p = \{\emptyset\} \\ \llbracket E \rrbracket_t^2 &= \emptyset \end{aligned}$$

Definition of the ports of the entity:

- $P^{in} = \{e_1\}$
- $P_E^{out} = \{e_2\}$
- $P_E^{internal} = \{\tau\}$

Definition of the variables of the entity:

$V_E = V_E^d \uplus V_E^p$, with

- $V_E^d = \{v_{email}, v_{credentials}, v_{http}, v_{messages}, v_{confidentialData}\}$, with, at the initial state
 - $v_{email} = \{\text{employeeEmail}\}$
 - $v_{messages} = \emptyset$
 - $v_{credentials} = \{\text{employeeCredentials}\}$
 - $v_{confidentialData} = \emptyset$
 - $v_{http} = \emptyset$
- $V_E^p = \{v_C \mid v_C \sim \mu \text{ s.t. } \mu(\tau.E) = n_1, \mu(e_2.E) = n_2\}$

Figure 4.13 illustrates the formal transformation we operated.

Communications between two IoT entities e_1 and e_2 in the IoT language are converted into a set of, possibly guarded, interactions between components \mathcal{B}_{e_1} and \mathcal{B}_{e_2} corresponding to e_1 and e_2 , respectively.

Definition 18 (Interactions for an IoT state). Let us consider two BIP atomic components B_{e_1} and B_{e_2} that are the respective transformations of two IoT entities: e_1

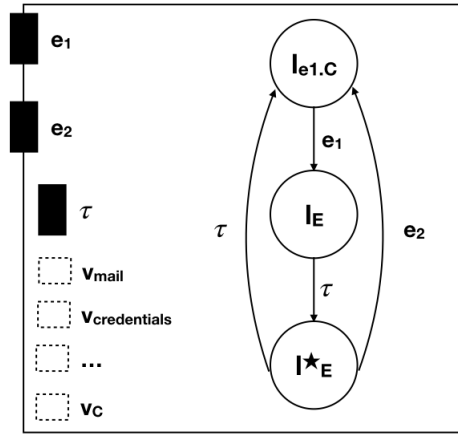


FIGURE 4.13: Illustration of the Formal Transformation of an Entity

and e_2 .

We denote the two atomic components $B_{e_1} = (P_1, V_1, \mathcal{N}_\infty)$ and $B_{e_2} = (P_2, V_2, \mathcal{N}_\infty)$. We write $\text{Actions} = \cup_{i \leq 2} P_i$, with $i \in I$, with I a set of indices such that $I = \{0, ..n\}$ for all the actions a of both BIP components.

We saw earlier in Chapter 3 that for an IoT system to be semantically correct, for each action in the system, there need to be a counterpart action.

We have that if $a \in \text{Actions}$, then if:

- $a = e_1 \xrightarrow[c]{v} e_2$ there is an action $a' \in \text{Actions}$ such that $a' = e_2 \xleftarrow[c]{v} e_1$,
- $a = e_1 \xrightarrow{v} e_2$ there is an action $a' \in \text{Actions}$ such that $a' = e_2 \leftarrow e_1$.

If for a given action, there is a counterpart action in the system, we define the interaction: $\gamma = (P, G, F)$ where:

- $P = \{a, a'\}$, represents the couple of actions;
- G is the guard of the interaction such that:
 - if the communication is based on a protocol (actions of type Send/Receive), the guard will check that the protocol conditions are met:
if $a = e_1 \xrightarrow[c]{v} e_2$ then $G = (\exists x \in v_c^1 \text{ s.t. } x \in v_c^2)$ for $v_c^1 \in V_1^d$, $v_c^2 \in V_2^d$
 - otherwise no protocol verification is required and $G = \text{true}$
- F is the update function of the transition. After a Send or Collect action, depending on the protocol associated to the variable, we add the variable to the knowledge of e_2 :
if $a = e_1 \xrightarrow[c]{v} e_2$ or if $a = e_2 \leftarrow e_1$,
then $F = \{v_{c'}^2 := v_c^2 \cup \{v\} \mid \text{protocol}(v) = c', v_c^2 \in V_2^d\}$

where V_1^d , V_2^d are the deterministic variables of \mathcal{B}_{e_1} and \mathcal{B}_{e_2} , respectively.

Atomic components can also carry out internal interactions that we define by:

$(\{\tau\}, \text{true}, F)$, where $F = \{v := v \mid v \in V^d\}$, for every component $\mathcal{B}_e = (P, V, N)$.

4.3.3 Bisimulation

In order to show a correspondance between the Security Oriented Modeling Language and the BIP language, we are going show first a correspondance in the transformation

from IoT to BIP and then in the transformation from BIP to IoT. In order to do that we need to define a few preliminaries concepts and notations.

First, we recall that a transition is defined, either for a pair of Send/Receive, Leak/-Collect interactions or for an Internal transition.

For an IoT system composed of n entities and of an initial state s_0 , we write Γ for the set of interactions between these entities as we did in Definition 11.

We write $\Gamma(\mathcal{B}_1, \dots, \mathcal{B}_n)$ the composition of n IoT components as we defined it in Definition 12 and $(\Gamma(\mathcal{B}_1, \dots, \mathcal{B}_n))$ the corresponding BIP system.

Theorem 1 (Bisimulation). We consider an IoT system of n entities with an initial state s_0 such that $s_0 = \langle P_1, k_1 \rangle | \dots | \langle P_n, k_n \rangle$, where $\langle P_i, k_i \rangle$ is the initial state of an entity $e_i, i \leq n$.

We denote B_{e_i} the BIP transformation of the IoT system, $\mathbf{X}_{\text{init}}^i$ the initial valuation of the variables V_i of the entity e_i and Γ its set of interactions.

For $\mathcal{M} = \langle Q, P, \pi, q_0 \rangle$ the semantics of the BIP counterpart model with the valuation $\mathbf{X}_{\text{init}}^1 \sqcup \dots \sqcup \mathbf{X}_{\text{init}}^n$, there exists $\mathcal{R} \subseteq S \times Q$ a symmetric relation such that, following the rules of the equivalence between two languages we consider that:

- For each initial state existing in the IoT model, there is only one equivalent initial state in the BIP model
 $(s_0, q_0) \in \mathcal{R}$;
- For each transition in the IoT system, there exists a corresponding transition in the BIP system
 if $(s, q) \in \mathcal{R}$ then for all $s' \in S$ and $s \xrightarrow[l]{[n]} s' \in T$ there exists $q' \in Q$ and $q \xrightarrow{\gamma} q' \in \pi$ with $\mathbb{P}(q \xrightarrow{\gamma} q') = n$ such that $(s', q') \in \mathcal{R}$.

Lemma 1. Any two congruent IoT states have the same transformation in SBIP systems.

Proof. We proceed by cases on the congruence relation. First consider the congruence relation on states: For the monoid laws on $|$, note that the transformation results in a set of atomic components and therefore the order of states in the parallel composition does not matter. In the case where processes are congruent, we distinguish two subcases:

- (i) Threads in a parallel composition translate into tuples of states in the transformation of a process (Definition 16) where the order of the states does not matter;
- (ii) For the rest we use the fact that inside an atomic component the states that have congruent labels are identified. \square

Theorem 1. Let e_1, \dots, e_n be n entities of an IoT system (S, L, T, s_0) with the initial states $\langle P_1, k_1 \rangle, \dots, \langle P_n, k_n \rangle$.

$\mathcal{B}_{e_i} = (P_i, V_i, N_i)$ with $N_i = (L_i, L_{i,0}, T_i)$, is the transformation of the current state of the entity e_i , for $i \leq n$.

Also let $V_i = V_i^p \cup V_i^d$. We write (Q, P, π, q_0) for the semantics of $\Gamma(\mathcal{B}_{e_1}, \dots, \mathcal{B}_{e_n}) = (\Gamma, \mathbf{V}, \mathbf{N})$ with $\mathbf{N} = (\mathbf{L}, \mathbf{L}_0, \mathbf{T})$. Lastly \mathbf{X}_{init} is the initial valuation.

To construct the relation $\mathcal{R} \subseteq S \times Q$ required by the theorem, we first set some notations and constraints below. Informally, these constraints establish the relation between the processes and knowledge functions in states of S and the markings and the valuations, respectively, in states of Q .

1. **Correspondance between Processes and Markings.** Let us consider two threads: T and U . Let us write m_T for the marking associated with T and defined as follows:

$$m_T(l) = 1 \text{ if } \ell(l) = T \text{ or } \ell(l) = U^*, U = [n_1]T_1 + \dots, \quad \text{for some threads } T \\ 0 \text{ otherwise}$$

where (P, V, N) and $N = (L, L_0, T,)$ is obtained as in Definition 16 and where $l \in \mathcal{L}$. For a process $P = T_1 \mid \dots \mid T_m$, let us write m_P for the marking associated with P and defined as $m_{T_1} + \dots + m_{T_m}$.

2. **Correspondance between Knowledge and the Deterministic Variables.**

From Definition 16 it follows that for each thread T_j in a process P_i we define the set $V_i^d = \{v_c \mid c \text{ is a protocol used in } T_j\}$. From Definition 17 then the set of variables of $P_i = T_1 \mid \dots \mid T_m$ is $\cup_{j \leq m} V_j = \{v_c \mid c \text{ is a protocol used in } P_i\}$.

Then, if \mathbf{X}_i the current valuation of entity e_i , we require that $\mathbf{X}_i(v_c) = k_i(c)$, for $i \leq n$, $c \in C$ and $v_c \in V_i^d$. Recall that we write C for the set of protocols used in the IoT system and k_i for the knowledge function of an entity e_i .

3. **Correspondance between Probabilistic Choices in Processes and the Random Variables.**

For every summation thread U in a process P_i , we have that there exists a random variable $v_U \in V_i^p$, by Definition 16. Moreover, if T a thread of P_i , belongs to a summation, i.e. $U = [n]T + T'$, for some threads T', U , then for the current valuation \mathbf{X}_i we have that $\mathbf{X}_i(v_U) = T$. For a process $P = T_1 \mid \dots \mid T_m$ we use Definition 17 and have that V^p is the disjoint union of all V_j^p , where V_j^p is the set of random variables for T_j , $j \leq m$.

We define the following relation between the states of S and the states of Q :

$$\mathcal{R} = \left\{ (\langle P_1, k_1 \rangle \mid \dots \mid \langle P_n, k_n \rangle, (m = m_{P_1} + \dots + m_{P_n}, \mathbf{X} = \mathbf{X}_1 \sqcup \dots \sqcup \mathbf{X}_n)) \mid \right. \\ \left. \text{the conditions 1-3 above hold} \right\}.$$

We show that \mathcal{R} is the relation required in Theorem 1. First we have to show that $(s_0, q_0) \in \mathcal{R}$.

We use Definition 16 from which we have that $L_0 \subseteq L$ is the initial place in the transformation of a thread T . Then, by Definition 17, $L_0 = \uplus_{j \leq m} \mathcal{L}_0^j = \uplus_{j \leq m} \{l_{T_j}\}$ is the initial set of places in the transformation of a process $P = T_1 \mid \dots \mid T_m$. From Definition 12 it follows that $\mathbf{L}_0 = \uplus_{i \leq n} L_{0,i}$. By Definition 10 the initial marking in $q_0 = (m_0, \mathbf{X}_{\text{init}})$ is defined as $m_0(l) = 1 \iff l \in \mathbf{L}_0$ and 0 otherwise. Hence we can write $m_0 = m_{P_1} + \dots + m_{P_n}$. This shows condition 1 of \mathcal{R} .

From Definition 17 we have that for each entity e_i , $\mathbf{X}_{\text{init}}(v_c) = k_i(c)$, for all protocols c used by e_i . From Definition 12 the set of variables of the composed \mathcal{B}_{e_i} components is the disjoint union V_i , i.e. $\mathbf{V} = \uplus_{i \leq n} V_i$, in particular $\mathbf{V}^d = \uplus_{i \leq n} V_i^d$. Then a valuation for \mathbf{V} is the disjoint composition of the individual valuations for V_i , from which it follows the required decomposition of \mathbf{X}_{init} in $q_0 = (m_0, \mathbf{X}_{\text{init}})$. Therefore condition 2 of \mathcal{R} holds. For condition 3 to hold it suffices to note that there is no probabilistic choice made yet in any process and therefore there is no correspondence to show. We can take any initial valuation we want for the random variables.

On the transformation from IoT to BIP

Let us now suppose that $(s, q) \in \mathcal{R}$ and that $s \xrightarrow[l]{[n]} s'$, for some label $l \in L$, some probability n and state $q' \in Q$. We have to show that there exists $q' \in Q$ and $q \xrightarrow{p} q' \in \pi$ with $\mathbb{P}(q \xrightarrow{p} q') = n$ such that $(s', q') \in \mathcal{R}$. We reason by cases on the label l of the transition $s \xrightarrow[l]{[n]} s'$.

- **In the case of an interaction between two entities**

Let e_1 and e_2 be the two communicating entities, and $l = SR : v$ or $l = LC : v$. We can write the transition as follows using Definition 10:

$$\begin{aligned} s &= \langle P_1, k_1 \rangle \mid \langle P_2, k_2 \rangle \mid \langle P_3, k_3 \rangle \mid \dots \mid \langle P_n, k_n \rangle \xrightarrow[l]{[1/m]} \\ s' &= \langle Q_1, k'_1 \rangle \mid \langle Q_2, k'_2 \rangle \mid \langle Q_3, k'_3 \rangle \mid \dots \mid \langle Q_n, k'_n \rangle \end{aligned}$$

where we can decompose $P_1 \equiv_P a_1.T_1 \mid P'_1$ and $P_2 \equiv_P a_2.T_2 \mid P'_2$, $Q_1 \equiv_P T_1 \mid P'_1$ and $Q_2 \equiv_P T_2 \mid P'_2$, again by Lemma 1 and from the rules of Figure 3.6. Here we suppose w.l.o.g. that a_1 and a_2 are the two synchronizing actions in P_1 and P_2 , respectively. Also suppose w.l.o.g. that a_1 is a send (or a leak) and that a_2 is a receive (or a collect). Let c be the protocol used for the communication in case $l = SR : v$.

From $(s, q) \in \mathcal{R}$ we have that $q = (m_{P_1} + \dots + m_{P_n}, \mathbf{X}_1 \sqcup \dots \sqcup \mathbf{X}_n)$ and that $m_{P_i} = m_{a_i.T_i} + m_{P'_i}$, for $i \leq 2$. Also from condition 1 of \mathcal{R} , $m_{a_i.T_i} = \{l_i\}$ with either $\ell(l_i) = a_i.T_i$, or $\ell(l_i) = U_i^*$, for some summation threads U_1, U_2 .

- If $\ell(l_1) = a_1.T_1$ then we use the transformation of Definition 16 to show that there exists the place $l'_1 \in L_1$, with $\ell(l'_1) = T_1$ and the transition $t_1 = (\{l_{a_1.T_1}\}, \langle a_1, g_1 = \text{true}, f_1 \rangle, \{l_{T_1}\})$ in \mathcal{B}_1 .
 - * If $\ell(l_2) = a_2.T_2$ then as above, there exists $l'_2 \in L_2$, with $\ell(l'_2) = T_2$ and the transition $t_2 = (\{l_{a_2.T_2}\}, \langle a_2, g_2 = \text{true}, f_2 \rangle, \{l_{T_2}\})$ in \mathcal{B}_2 .
 - * $\ell(l_2) = U_2^*$, with $U_2 = [n_2]a_2.T_2 + U'_2$, for some threads U_2, U'_2 . As in the case above, from Definition 16 we have that there exists the places $l'_2 \in L_2$ with $\ell(l'_2) = T_2$. We also have, from condition 3 of \mathcal{R} that there exists a random variable $v_{U_2} \in V_2^p$ with $\mathbf{X}(v_{U_2}) = a_2.T_2$. Moreover we have the transition $t_2 = (\{l_{U_2^*}\}, \langle a_2, g_2 = (v_{U_2} == a_2.T_2), f_2 \rangle, \{l_{T_2}\})$ in \mathcal{B}_2 .
- the other case is similar.

Note that in all cases above, $f_i = \{v := v \mid v \in V^d\}$ with $R_i^p = \emptyset$, $i \leq n$.

Using Definition 11 we have that there exists an interaction $\gamma = (\{a_1, a_2\}, G, F)$ such that

- If $l = SR : v$ then $G = (\exists x \in v_c^1 \text{ such that } x \in v_c^2)$ for $v_c^1 \in V_1$ and $v_c^2 \in V_2$.
- If $l = LC : v$ then $G = \text{true}$.

Also, $F = \{v_{c'}^2 := v_c^2 \cup \{v'\} \mid \text{protocol}(v') = c', v_c^2 \in V_2\}$ for both $l = SR : v$ and $l = LC : v$.

We now use Definition 12 and have that there exists the transition

$$\underline{T} = (\{l_1, l_2\}, \langle \gamma, g_1 \wedge g_2 \wedge G, (f_1 \sqcup f_2) \circ F \rangle, \{l'_1, l'_2\}) \in \mathbf{T}.$$

We have to show that the guard $g = g_1 \wedge g_2 \wedge G$ holds for the current valuation \mathbf{X} :

- If $g_1 = (v_{U_1} == a_1.T_1)$ then $\mathbf{X}(g_1)$ holds from condition 3 of \mathcal{R} ; otherwise $g_1 = \text{true}$. We proceed similarly for g_2 .
- If $l = SR : v$ then $G = (\exists x \in v_c^1 \text{ such that } x \in v_c^2)$ for $v_c^1 \in V_1$ and $v_c^2 \in V_2$. From condition 2 of \mathcal{R} we have that $\mathbf{X}(v_c^i) = k_i(c)$, $i \leq n$. Then the guard holds as it is the condition of rule SENDRECEIVE in Figure 3.6. If $l = LC : v$ then $G = \text{true}$.

Therefore, by Definition 10, there exists the transition

$$q = (m_{P_1} + m_{P_2} + \dots m_{P_n}, \mathbf{X}_1 \sqcup \dots \sqcup \mathbf{X}_n) \xrightarrow{\gamma} q' = (m', \mathbf{X}')$$

where we have to show that conditions 1-3 of \mathcal{R} hold. For condition 1 we have to show that $m' = m_{Q_1} + m_{Q_2} + \dots m_{P_n}$. Using Definition 10 it follows that

$$m' = m - \bullet \underline{T} + \underline{T} \bullet = m - \{l_1, l_2\} + \{l'_1, l'_2\}.$$

As $\mathbf{L}_0 = \uplus_{i \leq n} L_{0,i}$, from Definition 12, it follows that

$$m' = (m_{P_1} - \{l_1\} + \{l'_1\}) + (m_{P_2} - \{l_2\} + \{l'_2\}) + \dots + m_{P_n}.$$

Using condition 1 of \mathcal{R} on m_{P_1} and m_{P_2} we have that $m_{P_1} - \{l_1\} + \{l'_1\} = m_{Q_1}$ and similarly for m_{Q_2} .

Let us now show condition 2, i.e. $\mathbf{X}' = \mathbf{X}'_1 \sqcup \mathbf{X}'_2 \sqcup \dots \sqcup \mathbf{X}_n$ and $\mathbf{X}'_i(v_{c'}) = k'_i(c')$, $i \leq 2$. Using the function F above we have that $\mathbf{X}'_i(v_{c'}) = \mathbf{X}_i(v_{c'}) \cup \{v\}$. From rules SENDRECEIVE and LEAKCOLLECT we also get that $k'_i(c') = k_i(c) \cup \{v\}$, $i \leq 2$.

As $R_1^p = R_2^p = \emptyset$ condition 3 is trivial.

Lastly, the two transitions have the same probability: $|\text{Enabled}(m; \mathbf{X})| = m$, and therefore $\mathbb{P}(q \xrightarrow{p} q') = 1/m$.

• **In the case of an internal transition**

Let $l = \tau$; let e_1 be the entity that triggers the internal transition. Using Lemma 1 we can rewrite the states in the transition as follows:

$$\begin{aligned} s &= \langle P_1, k_1 \rangle \mid \langle P_2, k_2 \rangle \mid \langle P_3, k_3 \rangle \mid \dots \mid \langle P_n, k_n \rangle \xrightarrow[l]{[n]} \\ s' &= \langle Q_1, k'_1 \rangle \mid \langle P_2, k_2 \rangle \mid \langle P_3, k_3 \rangle \mid \dots \mid \langle P_n, k_n \rangle. \end{aligned}$$

There are two possibilities: either $P_1 \equiv_P \sum_{i \in I_1} a_i.T_i \mid P'_1$ where $Q_1 = a_1.T_1$ w.l.o.g. or $P_1 \equiv_P \tau.T_1 \mid P'_1$ with $Q_1 = T_1$. We write $U = \sum_{i \in I_1} a_i.T_i$ or $U = \tau.T_1$ depending on which of the two cases we are.

From $(s, q) \in \mathcal{R}$ we have that $q = (m_{P_1} + \dots m_{P_n}, \mathbf{X}_1 \sqcup \dots \sqcup \mathbf{X}_n)$ and that $m_{P_1} = \{l\} + m_{P'_1}$, $\ell(l) = U$. We use the transformation of Definition 16 to show that there exists the place $l' \in L_1$ and the transition $t = (\{l\}, \langle \tau, g = \text{true}, f \rangle, \{l'\})$ in \mathcal{B}_1 .

- If $U = \sum_{i \in I_1} a_i.T_i$ then $\ell(l') = U^*$, $f = \{v := v \mid v \in V_1^d\}$ and $R^p = \{v_U\}$.
- If $U = \tau.T_1$ then $\ell(l') = T_1$, $f = \{v := v \mid v \in V_1^d\}$ and $R^p = \emptyset$.

Using Definition 11 we have that there exists an interaction $\gamma = (\{\tau\}, G = \text{true}, F)$ with $F = \{v := v \mid v \in V_1^d\}$.

From Definition 12 there exists the transition

$$\underline{T} = (\{l\}, \langle \gamma, g_1 \wedge G = \text{true}, f \circ F \rangle, \{l'\}) \in \mathbf{T}.$$

The guard trivially holds and we obtain the transition

$$q = (m_{P_1} + \dots + m_{P_n}, \mathbf{X}_1 \sqcup \dots \sqcup \mathbf{X}_n) \xrightarrow{\gamma} q' = (m', \mathbf{X}')$$

where we have to show that conditions 1-3 of \mathcal{R} hold. As in the first case, condition 1 follows from $m' = m - \{l\} + \{l'\} = m_{Q_1} + \dots + m_{P_n}$. Condition 2 trivially hold as the update functions f and F are the identity and therefore $\mathbf{X}_1' = \mathbf{X}_1$. Indeed the knowledge function of k_1 is not modified by the rules CHOICE or INTERNAL.

To show condition 3 we use Definition 16 from which we have that there exists $v_U \in V_1^p$, $v_U \sim \mu$, where $\mu(a_1.T_1) = n_1$. Then we can take $\mathbf{X}'(v_U) = a_1.T_1$. We also this argument to show that the two transitions have the same probabilities: by Lemma 1, $|\text{Enabled}(m; \mathbf{X})| = m$ and therefore $\mathbb{P}(q \xrightarrow{p} q') = 1/m \times n_1$.

On the transformation from BIP to IoT

Hereafter we prove the similarity of the IoT system to its corresponding SBIP model. Let us suppose that $(q, s) \in \mathcal{R}$ and that $q \xrightarrow{\gamma} q'$, for a transition labelled with γ , where $q, q' \in Q$. We have to show that there is a state $s' \in S$ with $s \xrightarrow[l]{[n]} s'$, for some label $l \in L$, such that $(s', q') \in \mathcal{R}$. We define $s = \langle P_1, k_1 \rangle \mid \langle P_2, k_2 \rangle \mid \dots \mid \langle P_n, k_n \rangle$. Here we also reason by cases: whether the transition is an interaction between two components \mathcal{B}_{e_1} and \mathcal{B}_{e_2} or an internal transition.

- **In the case of an interaction**

We consider the communication is an interaction $\gamma = (\{a_1, a_2\}, G, F)$ between \mathcal{B}_{e_1} and \mathcal{B}_{e_2} :

$$q = (m_{P_1} + m_{P_2} + \dots + m_{P_n}, \mathbf{X}_1 \sqcup \mathbf{X}_2 \sqcup \dots \sqcup \mathbf{X}_n) \xrightarrow{\gamma} q' = (m', \mathbf{X}')$$

As it is an interaction between two entities, from Definition 12 we have that there exists the transitions $t_i = (m_i, \langle p_i, g_i, f_i \rangle, m'_i) \in T_i$, for $i \in \{1, 2\}$. From the Definition 11, $m_i = m_{P_i}$, $p_i = a_i$, $g_i = \text{true}$ and f_i are the constant update functions. From $(q, s) \in \mathcal{R}$ we have that $m_{P_1} = m_{a_1.T_1} + m_{P'_1}$, $m_{P_2} = m_{a_2.T_2} + m_{P'_2}$ with $P_1 = a_1.T_1 \mid P'_1$ and $P_2 = a_2.T_2 \mid P'_2$. Moreover, from the Definition 10 there exists the transition

$$\underline{T} = (m_{P_1} + m_{P_2}, \langle \{a_1, a_2\}, g_1 \wedge g_2 \wedge G, (f_1 \sqcup f_2) \circ F \rangle, m_{Q_1} + m_{Q_2}) \in \mathbf{T}$$

with $m_{Q_1} = m_{T_1} + m_{P'_1}$, $m_{Q_2} = m_{T_2} + m_{P'_2}$.

We distinguish between the two types of interactions:

- $a_1 = e_1 \xrightarrow[v']{c} e_2$ and there exists $a_2 \in \text{Actions}$ such that $a_2 = e_2 \xleftarrow{c} e_1$,
- or $a_1 = e_1 \xrightarrow[v']{\longrightarrow} e_2$ and there exists $a_2 \in \text{Actions}$ such that $a_2 = e_2 \leftarrow e_1$

Following the Definition 11 we have the following guards:

- if $G = (\exists x \in v_c^1 \text{ such that } x \in v_c^2)$ for $v_c^1 \in V_1$ and $v_c^2 \in V_2$ then $l = SR$
- if $G = \text{true}$ then $l = LC$

We can then apply the rules SENDRECEIVE or LEAKCOLLECT from Figure 3.6. Hence we derive an interaction between e_1 and e_2 exists for which we have to show that conditions 1-3 of \mathcal{R} holds.

$$s = \langle P_1, k_1 \rangle \mid \langle P_2, k_2 \rangle \mid \dots \mid \langle P_n, k_n \rangle \xrightarrow[l]{[n]} \\ s' = \langle Q_1, k'_1 \rangle \mid \langle Q_2, k'_2 \rangle \mid \dots \mid \langle P_n, k_n \rangle.$$

From above, it follows that $m' = m_{Q_1} + m_{Q_2} + \dots m_{P_n}$, which is the first condition of \mathcal{R} .

In the interaction γ , we apply the update function $F = \{v_c^2 := v_c^2 \cup \{v'\} \mid \text{protocol}(v') = c', v_c^2 \in V_2\}$ for both $l = SR : v$ and $l = LC : v$, then $\mathbf{X}'_i(v_c) = \mathbf{X}_i(v_c) \cup \{v\}$.

Therefore we can write $\mathbf{X}' = \mathbf{X}'_1 \sqcup \mathbf{X}'_2 \dots \mathbf{X}_n$. With the interaction $s \xrightarrow[l]{[n]} s'$, we apply rules SENDRECEIVE or LEAKCOLLECT from Figure 3.6 where $k'_i(c') = k_i(c) \cup \{v\}$. Hence the condition 2 holds, i.e. $\mathbf{X}'_i(v_c) = k'_i(c')$. With the execution of the γ interaction, the probabilistic distribution $R_1^p = R_2^p = \emptyset$, and from the SENDRECEIVE or LEAKCOLLECT from Figure 3.6 is the same, then the condition 3 trivially holds. The two transitions have the same probability: $\mathbb{P}(q \xrightarrow{p} q') = 1/m$, and therefore $|\text{Enabled}(m; \mathbf{X})| = m$.

- **In the case of an internal transition**

We consider the transition to be an internal transition τ in component \mathcal{B}_{e_1} . From lemma 1 we can write the transition:

$$q = (m_{P_1} + m_{P_2} + \dots m_{P_n}, \mathbf{X}_1 \sqcup \mathbf{X}_2 \sqcup \dots \sqcup \mathbf{X}_n) \xrightarrow{\gamma} q' = (m', \mathbf{X}')$$

where $s = \langle P_1, k_1 \rangle \mid \langle P_2, k_2 \rangle \mid \dots \mid \langle P_n, k_n \rangle$, from $(q, s) \in \mathcal{R}$, we distinguish two cases of transition execution:

- A probabilistic choice: $m_{P_1} = \{\ell\} + m_{P'_1}$ where $\ell(l) = \sum_{i \in I} [n_i] a_i.T_i$ and $P_1 = \sum_{i \in I} a_i.T_i | P'_1$. From the transformation of Definition 16, the transition:

$$t = (\{l_T\}, \langle \tau, \text{true}, f^* \rangle, \{l_{T^*}\}) \in T_1$$

can be executed where $f^* = (\{v := v \mid v \in V^d\} \text{ and } R^p = \{v_T\})$. From relations of Figure 3.6, there exists a CHOICE transition in IoT system such that

$$s = \langle P_1, k_1 \rangle \mid \langle P_2, k_2 \rangle \mid \dots \mid \langle P_n, k_n \rangle \xrightarrow[l]{[n_1]} \\ s' = \langle Q_1, k'_1 \rangle \mid \langle P_2, k_2 \rangle \mid \dots \mid \langle P_n, k_n \rangle$$

where $Q_1 = a_1.T_1$. Now we can verify if the conditions 1-3 of R holds. We have that $m_{Q_1} = \{\ell\}^*$ and $m' = m_{Q_1} + m_{P_2} + \dots m_{P_n}$. As the update function f is the identity function the condition 2 trivially hold and the knowledge $k'_1 = k_1$. To show condition 3, we note that there exists

$v_{T_1} \in V_1^p$, $v_{T_1} \sim \mu$ such that $\mathbf{X}'(v_{T_1}) = a_1.T_1$. We use Definition 16 from which we have that where $\mu(a_1.T_1) = n_1$.

- An internal transition: $m_{P_1} = m_{\tau.T_1} + m_{P'_1}$ and $P_1 = \tau.T_1|P'_1$. From the transformation of definition 16, the transition $\underline{T} = (\{l_{a.T}\}, \langle a, \text{true}, f \rangle, \{l_T\})$ can be executed where $f = (\{v := v \mid v \in V^d\})$ and $R^p = \emptyset$. From relations of Figure 3.6, there exists an INTERNAL transition in IoT system such that

$$s = \langle P_1, k_1 \rangle \mid \langle P_2, k_2 \rangle \mid \dots \mid \langle P_n, k_n \rangle \xrightarrow[l]{[n]} \\ s' = \langle Q_1, k'_1 \rangle \mid \langle P_2, k_2 \rangle \mid \dots \mid \langle P_n, k_n \rangle$$

where $Q_1 = T_1|P'_1$. Now we can verify if the conditions 1-3 of R holds. We have that $m_{Q_1} = m_{T_1} + m_{P'_1}$ and $m' = m_{Q_1} + m_{P_2} + \dots m_{P_n}$.

As the update function f is the identity function the condition 2 trivially hold and the knowledge $k'_1 = k_1$. Then $\mathbf{X}' = \mathbf{X}'_1 \sqcup \mathbf{X}_2 \sqcup \dots \sqcup \mathbf{X}_n$. Likewise, since $R^p = \emptyset$ the condition 3 trivially holds.

□

4.4 Attack Tree to BIP

The attack tree describes the attack implementation by the attacker. It includes all the possible ways to reach the goal of the attack as, and for each way, all the steps to conduct the attack.

We include the attack tree in our methodology as a way to **monitor** the attack.

We specify it in the form of a json file, containing a certain number of information as we can see in Subsection 4.4.1. The parser transforms the json file into a BIP component that, without taking part in the execution, helps us following its progress, as we describe in Subsection 4.4.2.

4.4.1 Attack Tree Implementation

To describe attacks tree we store it in a json file which is called *textural representation*. The json format, which we have an example in Figure 4.14, provides the following information:

- the name of the node
- the type of the node
- the children nodes if the node is internal

If the node is an internal node, its name is the one we assigned to the subgoal and its type value is either "AND" or "OR". If the node is a leaf it represents an action possibly taken by the Attacker. We consider the actions to be either sending or receiving information through a protocol, either leaking or collecting information. As we have seen in Definition 3 the name of the leaf is the value of the exchanged information and is either "SR" for Send/Receive, either "LC" for Leak/Collect. Therefore it is possible to monitor what data the Attacker had access to and by what kind of communication it happened. Using our running example in 4.15, let's see how the attack tree is represented. The Attacker needs to collect the employee credentials in order to connect to the server and collect the sensitive information stored on it. The

```

{
  "name" : "hack_server",
  "type" : "AND",
  "children" :
  [
    {
      "name" : "employeeCredentials",
      "type" : "LC"
    },
    {
      "name" : "secretInformation",
      "type" : "LC"
    }
  ]
}

```

FIGURE 4.14: Running Example: Attack Tree JSON File

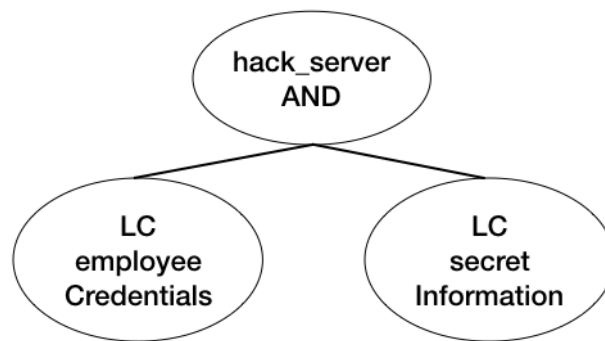


FIGURE 4.15: Running Example: Attack Tree

left leaf of the tree represents the employee leaking his credentials. The value of the leaf is "employeeCredentials" which is the leaked value and its type is "LC". The same goes for the right leaf that represents the leak of the server private information. Another case that is not represented here would be to have a "SR" type of node which would mean that the value gathered by the Attacker was received through a safe communication. The root of the node is an "AND" type of node, which means that in order for the attack to be successful, the Attacker must have collected both the employee credentials and the server secret information.

4.4.2 BIP Monitor

The monitor is turned into an atomic BIP component by the parser. This component that we can see an example of in Figure 4.18, contains only one state but a variety of transitions. For each step of the attack tree the monitor contains one transition. The transition is fired when the attacker makes the corresponding action. Each step of the attack is also represented in the monitor via a boolean value that indicates whether or not the attacker made the action.

In addition, the monitor contains a score variable. This variable is a float which value is initialised to "0". Throughout the actions of the attacker, the value is incremented so it matches how "far" the attack is. When the score value is equal to the height of the tree, it means that the root of the tree has been reached and that the attack has been successfully conducted. During a transition the monitor does the following operations:

- the corresponding boolean variable is updated to "TRUE"

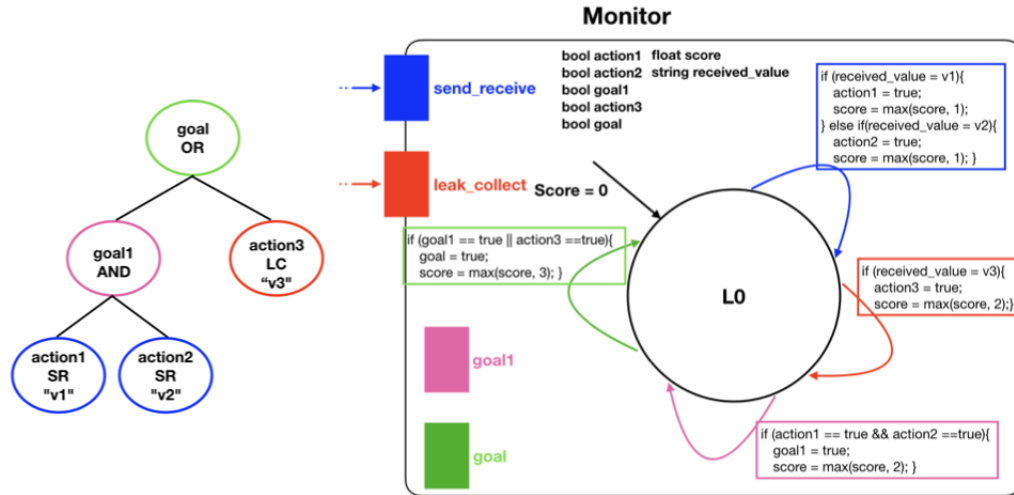


FIGURE 4.16: Correspondance between Monitor and Attack Tree

- the monitor checks if this boolean variable is connected in the tree to an internal node of type "AND" or "OR"
 - if the parent node is of type "OR", then the boolean variable that represents the internal node is updated to "TRUE"
 - if the parent node is of type "AND", and all other sibling nodes' value is "TRUE" then the boolean variable is updated to "TRUE", otherwise its value remains the same
- the variable score is incremented, depending on the height of the node in the tree (see Definition 4)

To perform these variable updates, the monitor has to know when the attacker is taking an action and what is the action. To be aware of the execution of the attack, we need to connect the monitor and the model somehow, without the monitor being able to influence the running of the model in any way.

We recall that the components of the model communicate with each other using their exported ports which are connected by different kinds of connectors. To export the actions of the components outside of the model, we add a port on the connectors, that can be exported outside of the model. This allows the connector to export information about the communication going on. On the monitor side, we add an exported port on each of his transitions. Indeed, the monitor contains 2 exported ports: one for the interactions of type Leak/Collect, and another one for the interactions of type Send/Receive. This way we can connect the transitions of the monitor to the information received from the outside.

The connection between the model and the monitor is realized in a third BIP file called "model_monitor". In the file we declare a type of connector that can connect the exported ports of the model and the exported ports of the monitor, and that transmits information about the action performed in the model to the monitor. This way, the monitor is in a passive position: he only receives information, without having the possibility to interfere with the model operations. The BIP file also includes an instance of the model and one of the monitor and is used as an entry point of the execution.

Figure 4.16 shows a concrete example of the correspondance between the monitor and the attack tree. We can see that the internal nodes "goal1" and "goal" are represented as internal transitions in the monitor. We can also see that for each action taken by the attacker that corresponds to a node in the tree, the score variable is incremented. It is important to notice that the value of the score corresponds to the height of the highest node the attacker is at. For instance if the attacker has successfully performed "action3", the score value is "2", which is the height of the corresponding node. When the attack is done, the score value will be equal to the tree height.

4.4.3 Attack Tree Formal Transformation

Here, we formally define the BIP monitor as well as its creation from the attack tree. We recall that the leaves of the attack tree contain possible actions that the attacker can take to conduct an attack and the internal nodes define how these actions must be combined.

Considering an attack tree t composed of n nodes (including the leaves and the internal nodes). In the monitor built from t , for each node, we create a boolean variable denoted v_n . The value of each variable v_n is updated depending on the actions taken by the attacker.

For the tree t , we denote $h(t)$ its height and $d(n, t)$ the depth of the node n , that we defined in Definition 4. The monitor also contains a variable *score* that keeps up with the progress of the attacker depending on his actions. This variable has a maximum value of $s_{max} = h(t)$ that can be reached or not during an execution according to the success of the attack. The score of an execution is written $s = h(t) - d(n, t)$, where n is the highest node for which v_n is true.

Definition 19 (Monitor). The monitor M_t of an attack tree t , is a BIP component $M_t = (P, V, N)$ where:

- $P = \{p_{SR}, p_{LC}, \cup_{i \leq n} p_i\}$ consists of two external ports: one for the Send/Receive and one for the Leak/Collect interactions, that are input ports, as the monitor only receives information. In addition, each internal node of the tree leads to the creation of an internal port.
- $V = V^d \cup V^p$, where $V^d = \{\text{score}\} \cup \{v_n | n \text{ is a node of } t\}$ and $V^p = \emptyset$
- $N = (\{l_o\}, \{l_o\}, T)$ is a Petri Net with only one place and with $T = \{ \{(l_o, \langle p, \text{true}, f \rangle, l_o) | p \in \{P_{SR}, P_{LC}\}\} \cup \{(l_o, \langle p, g, f \rangle, l_o) | p \in \cup_{i \leq n} P_i\} \}$

The interactions between the monitor and the system have as unique target to allow the monitor to observe the attacker behaviour and keep traces of the system state. From the information its receives, the monitor updates the value of the boolean variables for them to reflect the progress of the attack. All ports, whether they are internal or external update the score variable depending on their attributed height.

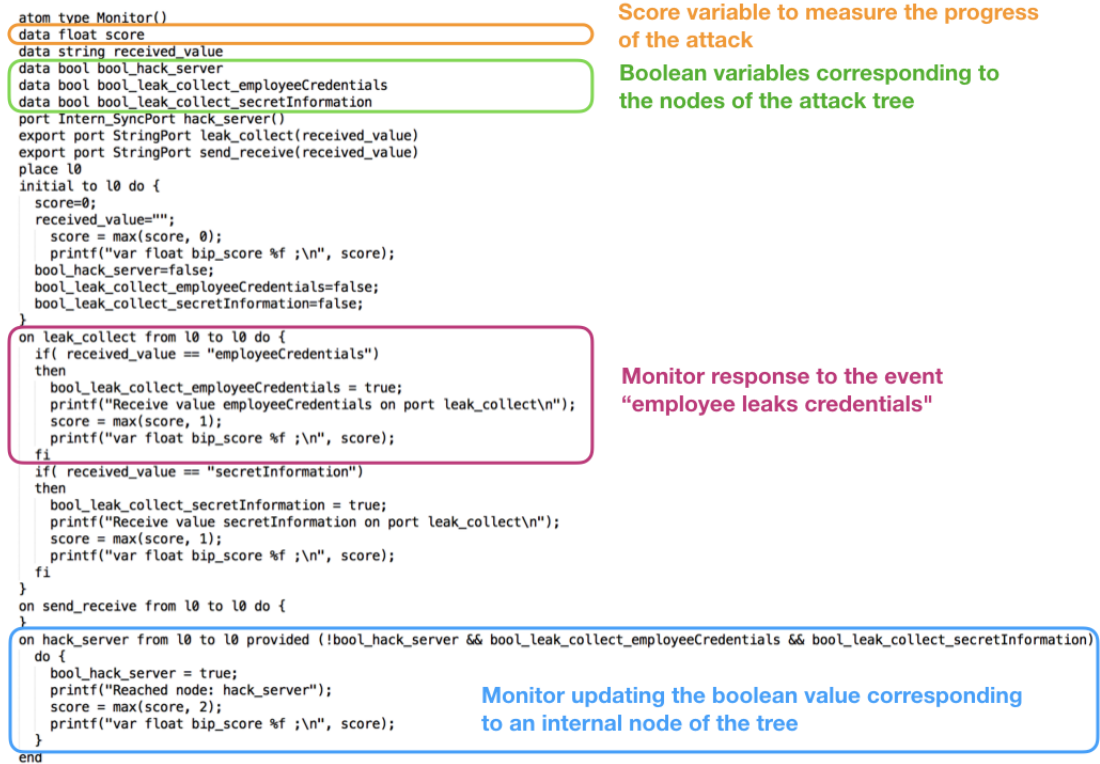
Definition 20 (Interaction between the monitor and the BIP system). Let Γ be a set of interactions of a BIP system. We define Γ' the set of interactions between the system and the monitor $M_t = (P, V, N)$.

For all interaction $\gamma = (\{a, a'\}, G_\gamma, F_\gamma) \in \Gamma$, happening in the system, we define a corresponding interaction between the system and the monitor: $\gamma' = (\{a, a', p\}, G_\gamma, F_\gamma \sqcup f) \in \Gamma'$ where:

- if $a = e_1 \xrightarrow[c]{v} e_2$, then $p = p_{SR}$ and $f = \{v_n := \text{true} | v = v'\}$
- $a = e_1 \xrightarrow[e]{e} e_2$, then $p = p_{LC}$ and $f = \{v_n := \text{true} | v = v'\}$

4.4.4 Running Example Monitor

The monitor is a BIP atomic component, contained in a separate BIP file from the model. We can see the BIP code of the monitor component in Figure 4.17 and its graphical representation in Figure 4.18.



```

atom_type Monitor()
data float score
data string received_value
data bool bool_hack_server
data bool bool_leak_collect_employeeCredentials
data bool bool_leak_collect_secretInformation
port Intern_SyncPort hack_server()
export port StringPort leak_collect(received_value)
export port StringPort send_receive(received_value)
place l0
initial to l0 do {
  score=0;
  received_value="";
  score = max(score, 0);
  printf("var float bip_score %f ;\n", score);
  bool_hack_server=false;
  bool_leak_collect_employeeCredentials=false;
  bool_leak_collect_secretInformation=false;
}
on leak_collect from l0 to l0 do {
  if( received_value == "employeeCredentials")
  then
    bool_leak_collect_employeeCredentials = true;
    printf("Receive value employeeCredentials on port leak_collect\n");
    score = max(score, 1);
    printf("var float bip_score %f ;\n", score);
  fi
  if( received_value == "secretInformation")
  then
    bool_leak_collect_secretInformation = true;
    printf("Receive value secretInformation on port leak_collect\n");
    score = max(score, 1);
    printf("var float bip_score %f ;\n", score);
  fi
}
on send_receive from l0 to l0 do {
on hack_server from l0 to l0 provided (!bool_hack_server && bool_leak_collect_employeeCredentials && bool_leak_collect_secretInformation)
do {
  bool_hack_server = true;
  printf("Reached node: hack_server");
  score = max(score, 2);
  printf("var float bip_score %f ;\n", score);
}
}
end

```

Score variable to measure the progress of the attack

Boolean variables corresponding to the nodes of the attack tree

Monitor response to the event "employee leaks credentials"

Monitor updating the boolean value corresponding to an internal node of the tree

FIGURE 4.17: BIP Code of the Monitor

We recall that the attack tree of the running example, displayed in Figure 4.14 and in Figure 4.15 contains three nodes, in which one is internal. Once the monitor receives a notification of a successful interaction of any type happening, it will check the value received by the attacker. If this value matches the value indicated by the attack tree, it has the indication that a step of the attack has been successfully conducted and this will have an effect on several monitor variables as we can see below.

We can see in Figure 4.17 that the BIP monitor also contains three boolean variables, one for each node of the attack tree. On the initial state, before the attack can begin, they are set to "FALSE". Then, for the two variables matching the actions of the attacker, their value are set to "TRUE" when the attacker has carried them out successfully. In the running example, the attack is a success if both the employee and the server have leaked information.

Once both of the actions are recognised as successful, it is necessary for the monitor to also update the boolean variable corresponding to the internal node of type "AND". To do so, the monitor checks that both boolean variables of the actions are true, and then set the third variable to "TRUE". In the case of an "OR" type of internal node, only one of the variable corresponding to a subnode would need to be of value "TRUE".

In addition to updating boolean variables, for each action, the monitor updates the

score variable. To each node of the tree, there is an associated value, corresponding to his height in the tree. In the running example, both leak actions have a value of "1" and the internal node's value is "2". So when the score's value matches the height of the attack tree (here the maximum value is "2"), then we know that the attack was completed.

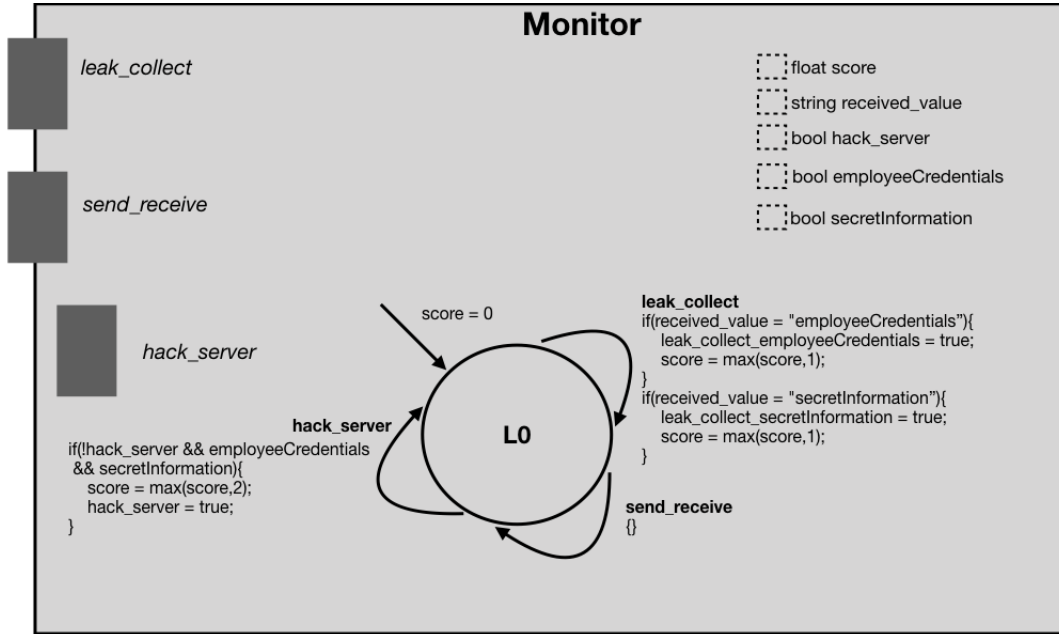


FIGURE 4.18: BIP component of the Monitor

Now let us see how the monitor is connected to the rest of the model. The monitor has two exported ports: one for the Send/Receive communications and another one for the Leak/Collect communications. This is how it is aware of the actions of the system. Let us focus now on how the model transmits these information to the monitor.

On each Send/Receive or Leak/Collect type of connector we add a port. This port takes a variable as a parameter. This variable is initialized using the value exchanged in the communication. This is how we can access, in the monitor, the value sent during the exchange. Figure 4.19 shows an example of the BIP code of a connector with an exported port. The connector declaration of Figure 4.19 allows the communication

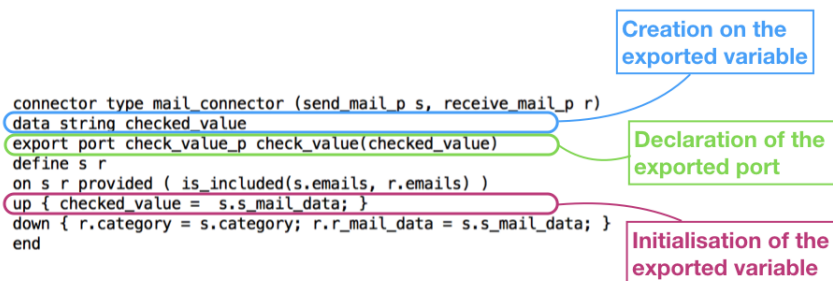


FIGURE 4.19: Example of a Connector with an Exported Port

using the protocol "mail". We add to it the port "check_value" that takes a variable "checked_value" as a parameter. The variable "checked_value" is initialized using the value exchanged via the "mail" protocol in the "up" part of the connector.

Finally, the actual connection between the model and the monitor is done in a third file, which we illustrate in a simplified way in Figure 4.20. For readability reasons, in Figure 4.20 we only represent one send and collect port in the attacker component when, in reality there is one external port for each action of this component. We also chose to not represent the internal ports as they are not part of the communication between the model and the monitor.

The third file includes an instance of the monitor, an instance of the model as well a connector declaration. The type of connector created makes the monitor able to connect the exported port of the model and the ports of the monitor.

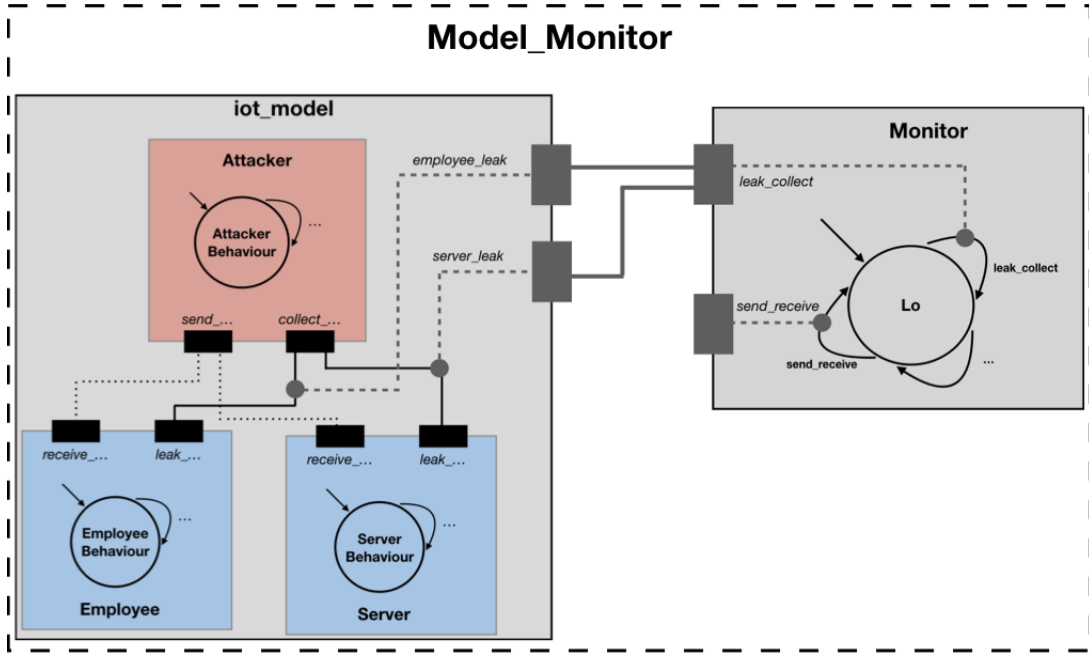


FIGURE 4.20: BIP component of the Monitor

4.20, we can see two exported ports on the model that are placed on the interactions that are described in the attack tree. These actions are the Leak/Collect interactions between the employee and the attacker and between the server and the attacker. We can also see that these interactions correspond to transitions in the monitor.

Chapter 5

Experiments

This last chapter contains concrete applications of our methodology through two use cases.

We first start in Section 5.1 by a brief recap of the process that lead us to the statistical analysis. We then delve into the statistical modeling by explaining the two algorithms that we use to perform the analysis: Monte Carlo and Importance Splitting.

The two following sections develop the concrete use cases we chose to test the methodology. Section 5.2 presents a Smart Hospital and Section 5.3 the Amazon Key system. Each section is constructed as follows.

They start with a presentation of the system: its nominal behaviour and of all of its actors using SOML. Next, we describe possible attacks towards the systems and their representations with an attack tree, as well as the corresponding BIP model. Finally, we show and comment the analysis results.

5.1 Methodology

Before presenting the actual results that we obtained with our models, let's introduce the methodology we followed to conduct the analysis. This section is composed of two parts. First, in Subsection 5.1.1, we explain what Statistical Model Checking is and the two algorithms we are using: Monte Carlo and Importance Splitting. Next, in Subsection 5.1.2, we start by a brief summary of the overall process and present the different parameters we chose for the analysis. This section also includes a presentation of the tools we used for the experiments.

5.1.1 Statistical Model Checking

To conduct our experiments, we used Plasma Lab, which is a Statistical Model Checker (SMC). Several algorithms are implemented in Plasma Lab and the ones that serves our needs are Monte Carlo and Importance Splitting.

Monte Carlo

The Monte Carlo method allows us to estimate the probability to satisfy a requirement. To conduct the experiments we need to specify the property to verify and the number of simulations that we want to run. Here the requirement is the success of the attack. The score of the monitor must then be equal to the height of the tree, as the root node represents the attack success. The probability of success is computed from the ratio of successful simulations compared to the total number of simulations.

Importance Splitting

Importance splitting is a method designed to calculate the probability of rare events to happen, Importance splitting decomposes the verification of the property into several properties that define a set of levels that a trace need to satisfy before the main property is satisfied. In our work, the property is the success of the attack that is represented by the root node of the attack tree. The subproperties that needs to be verified are the steps of the attack and therefore the nodes and the leaves of the attack tree.

The importance splitting algorithm [27] estimates the probabilities of passing from one level to the next by the proportion of a constant number of simulations reaching the upper level from the lower. The failing simulations are replaced by new simulations and are started from the states chosen uniformly from the terminal states of successful simulations. Figure 5.1 illustrates the process. The overall estimate is the product of the estimates of going from one level to the next.

We can express the overall estimate this way:

We consider γ to be the probability for an arbitrary execution trace to verify the temporal logic property φ .

We have: $\gamma = \prod_{i=1}^n P(l > l_i | l > l_{i-1})$ for the probability of reaching level l_i , with l being an abstract level of a path. For more details about the importance splitting process and algorithm, please refer to [33].

To conduct the experiments using the importance splitting algorithm with Plasma lab, we need to use a RML Observer (RDF Mapping Language). The observer will specify the requirement for the simulation trace to be successful. The property to verify is expressed using BLTL (Bounded Linear Temporal Logic). Plasma Lab includes a tool that transforms the BLTL property into a RML observer. As in the Monte

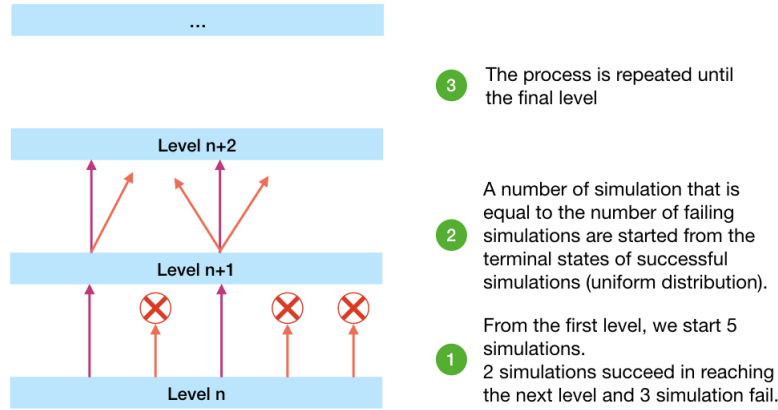


FIGURE 5.1: Importance Splitting algorithm

Carlo algorithm, we specify the success property (the score of the monitor equals the height of the associated attack tree) and the number of simulation that we want to run.

Concerning the practical aspect, we used Plasma lab (Platform for Learning and Advanced Statistical Model checking Algorithms) to perform the statistical analysis. It was introduced in [32] as a flexible SMC platform that included a built-in compiler and a virtual machine for the simulations. At its creation, it was able to perform rare event analysis, using the importance sampling algorithm. Afterwards it was upgraded with different model checking modes [10] such as Mont Carlo and importance splitting. The current Plasma Lab architecture [44], that we can see in Figure 5.2 enables Plasma Lab to perform model checking using external simulators.

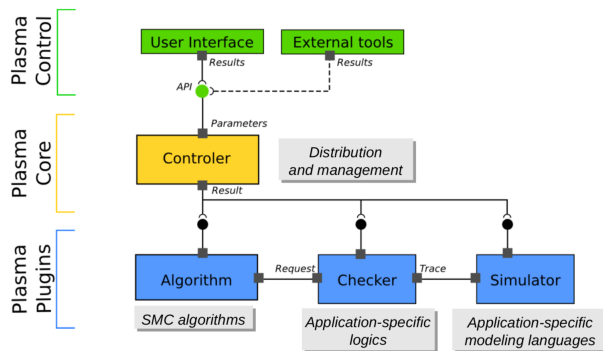


FIGURE 5.2: Plasma lab Architecture

5.1.2 Overview

Figure 5.3 shows a representation of the whole process from the inputs to the final results of the statistical analysis. As we recall from the previous chapters, the first step is to model a system, using SOML. The second step is to model different attacks, which have a common goal and could happen on the system using an attack tree. These two files represent the inputs of the process as we can see in Figure 5.3. Then, in order to perform the simulations used for the analysis, we transform the model as well as the attack tree into a BIP model. This is made possible thanks to the parser that we implemented. The SOML model is converted into an equivalent

BIP model, and the attack tree into a BIP monitor that observes the behaviour of the model during the simulation and keeps track of the attack happening. A third BIP file is generated in order to connect the model to the monitor. This way, the monitor can receive updates from the model.

The simulations are carried out by the BIP Engine. Throughout this process, the BIP Engine exchanges information with *Plasma Lab* which is the *Statistical Model Checker* we use.

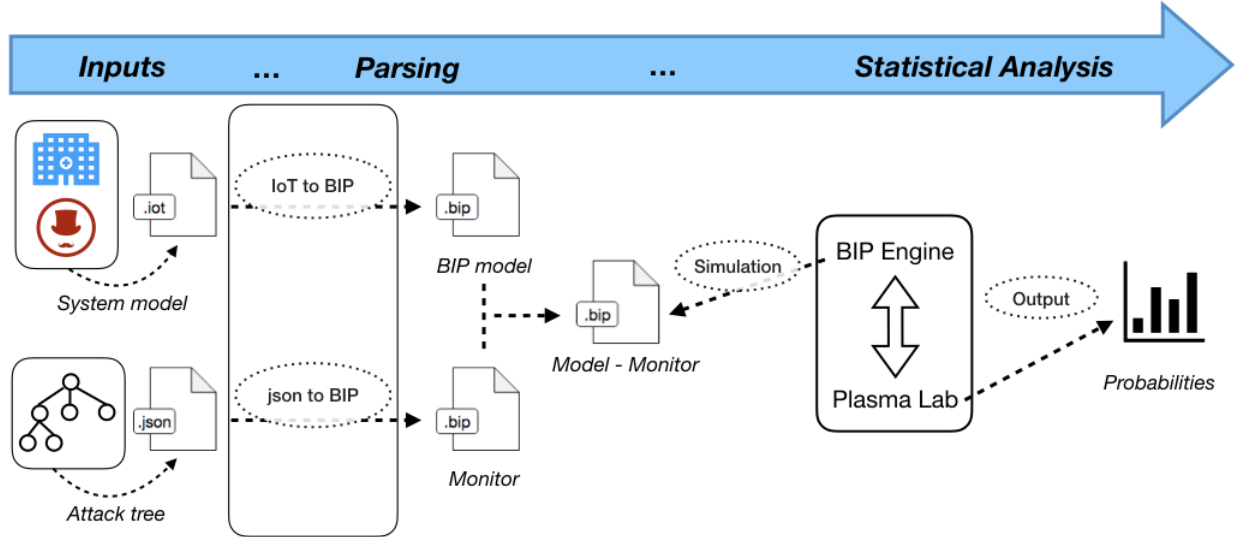


FIGURE 5.3: Process Overview

To be able to use Plasma lab, we developed a plugin that allowed us to connect the Plasma checker and the BIP engine as a simulator.

As we described in the previous subsection, SMC requires a property to verify. The probabilities computed are the chances for the system to verify this property. The property that we used for our simulations concerns the score of the monitor. As we know, from Subsection 4.4.3, the BIP monitor has a score variable. This score is incremented, depending on the node of the attack tree reached by the attacker during the simulations. When the monitor score variable reaches its highest value, it means that the attack has succeeded. Therefore, we specify as a property to the model checker that the monitor score must reach a certain value, corresponding to its highest possible score.

Concerning the conduct of the experiments, the results that we can see in Figure 5.8 and Figure 5.14 were obtained by taking the average of ten iterations for each model, for optimal reliability.

Between the different cases, the two varying parameters were the probabilities in the model and the number of simulations.

The probabilities, unlike the model and the attacks, are not based on real life examples. We simply increased the chances of failure for the attacker. At a certain point, the success of the attack becomes rare enough for it to be considered a rare event. For this purpose, anytime an entity had an action of leaking or sending information to the attacker that could benefit his attack, we decreased the chances for this entity to take this action. For instance, let's consider that the attacker sends an email to an employee, requesting his password. The employee has two choices: either he leaks

his password, or he just ignores the email which is modeled by an internal action. To decrease the chances for the attacker to access the password, and therefore to be one step closer to carrying out his attack, we put a higher weight on the internal action. We start both experiments with a model without any probabilities and therefore a uniform choice between all possibilities. Then, we assign a weight of 1 to the leak actions, or other actions of the internal entities that benefit the attacker. Throughout the experiments, we increase the weight on the internal counterpart actions that are meant to make the attacker fail. On the result tabs, we indicate the values of the weights on the internal actions for each experiment.

Since we want to know the probabilities for an attack to succeed as well as compare the efficiency of the Monte Carlo and Importance Splitting algorithms, we ran an increasing number of simulations for each case. This way we could see for each algorithm the minimum number of simulation necessary to give us a precise result as well as obtain the actual result.

By increasing the number of simulation on both algorithms, we also increased the necessary resources to conduct the experiments. We ran the simulations on the cluster from the Igrida Grid of INRIA Research Center Rennes Bretagne Atlantique. We allocated a node with 20 Gb of RAM.

The simulations were run using the 2018 version of the BIP engine.

5.2 Smart Hospital

This section describes the Smart Hospital use case. We first give a brief description of the system and its general concepts in Subsection 5.2.1. Subsection 5.2.2 shows the SOML model implemented for the experiments, and Subsection 5.2.3 the possible attacks towards it. Then, in Subsection 5.2.4, we introduce the BIP model built from the SOML model. Finally, Subsection 5.2.5 shows the results of the statistical analysis on the model.

5.2.1 Overview

IoT systems are used to push the boundaries of the hospital. Connected medical devices allow the medical staff to treat the patients remotely and give the staff real time access to their metrics. As the hospitals expand their limits, they increase their attack surface and are therefore more vulnerable. The migration from a hospital to a smart hospital usually happens by adding devices and features to the existing system, and often the security and the infrastructure are not updated. This lack of security upgrade and adaptation leaves the system vulnerable. In addition to that, the hospital staff who operate on the IT System on a daily basis are not sensitized to the risks.

5.2.2 Smart Hospital SOML model

The Smart Hospital example is constituted of four internal entities, and two external entities that conduct an attack.

The internal entities are the following:

Hospital IT	It is part of the interconnected Hospital information system. It provides an interface for employees and patients. Users can connect to it and access different services and information. For instance, the hospital staff can create a ticket if they encounter technical issues, and patients can update their data.
Hospital DataBase	It contains the information the Hospital needs to function properly. These information can be confidential (patient data, passwords for example) or not.
Employee	He is part of the Hospital staff. He can be a caregiver or not. He has credentials that give him access to critical information stored in the Hospital DataBase. He has a phone number and an email address which allow him to be contacted.
Medical Device	It can be an implantable, wearable networked device. It allows the hospital staff and systems to access real time measurements of key vital signs. Information is transmitted via network connections. They also allow medical staff to perform remote actions, for example administer a medical dose.

Two external entities perform an attack towards the system:

Attacker	This entity communicates with every internal entity he can reach in order to steal information. To this end, every existing protocol can be exploited.
Malware	This entity is a tool used by the attacker to infiltrate the system. Each information the malware accesses is transmitted to the attacker.

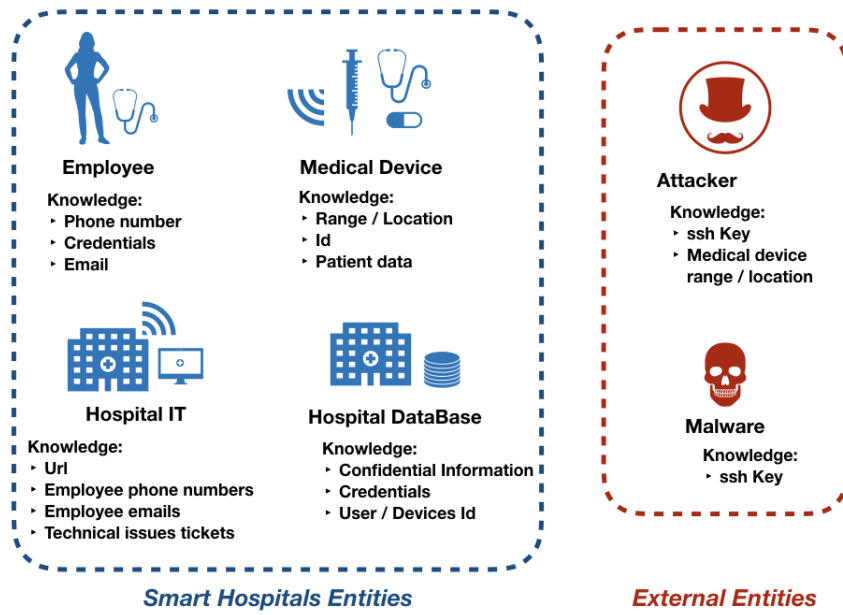


FIGURE 5.4: Initial State of the Smart Hospital System

Figure 5.4 represents the initial state of the system: all entities and their initial knowledge. Amongst other things, these data allow the entities to communicate with each other through protocols. We distinguish eight distinct protocols in the system:

- **url**: this protocol is used when communicating with the Hospital IT. The verification is made on the url of the Hospital IT.
- **phone**: entities can communicate with each other using their phones. In order to call someone it is necessary to know the receiver phone number.
- **ssh**: the attack and malware send each other information using the ssh protocol. To access this channel, one must have the ssh key.
- **https**: represents the secure connection with the server. A successful communication with the server is only possible for users with valid credentials.
- **mail**: a possible way for entities to communicate with each other is through emails. The verification is performed on the email addresses.
- **malware installation**: one of the attack includes the attacker installing a malware on an employee computer. In order to do that the employee must accept the installation, which can happen when he doesn't realize what he is actually agreeing to. This protocol represents the communication between the attacker and its employee target. The verification is performed on the email address of the target.
- **bluetooth**: the medical devices send and receive metrics via unprotected bluetooth. In order to connect to it, it is only necessary to be in the emission range of the device. This protocol then checks the physical location of the entity that tries to access it.
- **device communication**: medical devices transmit the patient metrics to the hospital database through this protocol. This protocol checks the device identification number before allowing an exchange.

5.2.3 Smart Hospital under Attacks

Here, we model several attacks towards the Smart Hospital. All these attacks are conducted with the intent of accessing the confidential data of the Hospital DataBase. The steps composing the attacks are represented in the attack tree of the Figure 5.5.

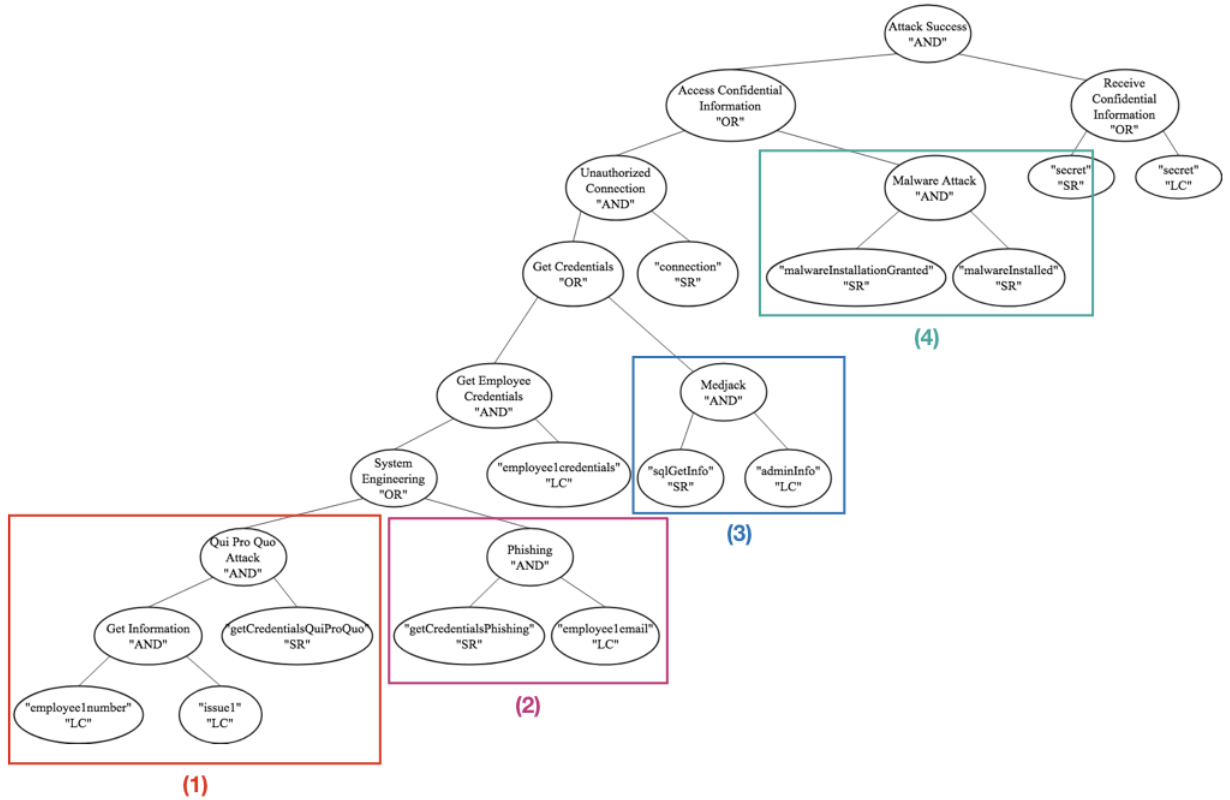


FIGURE 5.5: Attack Tree Smart Hospital

For this attack scenario, we drew on a number of existing attacks towards hospitals, described, in particular on the ENISA report [21] and on the TrapX reports [41]. To access the confidential data, we considered two possibilities. First, the attacker starts by gaining access to credentials and then can connect to the DataBase (1, 2 and 3). A second possibility is that the attacker hacks into the DataBase and gets direct access to the data (4). The rest of this section details the stages of each attack.

Qui Pro Quo Attack In this attack, represented in part (1) of the attack tree, the attacker gathers general information: employees phone numbers and technical issues that affect employees. Using these, the attacker calls employees, posing as technical support and ask them for their password. If an employee leaks his credentials on the phone, the first part of the attack is a success.

Phishing Attack This a well known attack that we show in part (2) of the attack tree: the attacker gets access to email adresses of the hospital employee. He sends spam emails, requesting the user to click on a link and then connect, using their password. This leads, once again, to the leak of their credentials.

Medjack Medjack is a new term to describe the attack through the hack of connected medical devices: it is the contraction of "medical" and "hijack" and it is shown in part (3) of the attack tree. It appeared for the first time in the reports of the TrapX Security organization [40]. In this case, the attacker takes control of a medical device that sends patient data via bluetooth. The attacker replaces the medical data by a SQL Injection. SQL (Structured Query Language) is a language designed for managing data held in a database. A SQL injection is a technique in which malicious SQL statement is inserted into an entry field for execution. So, instead of sending regular metrics, the device transfers a SQL request to the database in charge of collecting patient data. The SQL statement requests the display of all passwords, including the admin one. If the database is not protected against SQL Injections, it will execute the request and leak the passwords.

These three attacks have one thing in common: they represent a first step towards gaining access to the confidential information of the hospital database. For the attack to be a success they now need to connect to the server, using the stolen credentials. There is a risk that the password is wrong or does not provide enough clearance to the Attacker for him to access the information he seeks.

Malware Attack This attack allows the Attacker to directly access confidential information. As we can see in part (4) of the attack tree, the attacker starts by sending an email infected with a malware to an employee. If the employee opens the email, his computer which is connected to the network, can be infected. Once installed, the malware moves through the network, until it reaches the confidential information. These information are then transmitted to the Attacker by the malware via their ssh connection. For simplicity reasons, alternative steps of the attack such as parallel moves (steps where the malware moves through the system and makes information requests to the different entities it infects until it finds interesting information) of the malware within the IT system do not appear in the model.

5.2.4 Smart Hospital BIP Model

Once we have modeled the SOML model as well as the attack tree, there are both transformed into a BIP model in order to perform the simulation. Figure 5.6 shows the BIP model of the hospital. For readability reasons, the data of each component are not showed in the figure.

First, we can see that each SOML entity has a corresponding BIP component. In the transformation, each action of the entity leads to the creation of a unique port in the BIP model. In Figure 5.6 only one port for each protocol used by the entity is represented, for clarity purpose. For the same reason we only represent external ports and ignore the internal ones.

The BIP components' behaviours are represented using Petri Nets, and we can see that the entity and its matching BIP component behave in the exact same way.

Figure 5.7 shows a part of the BIP monitor generated from the attack tree of Figure 5.5. The figure focuses on the Phishing part of the attack, described previously in

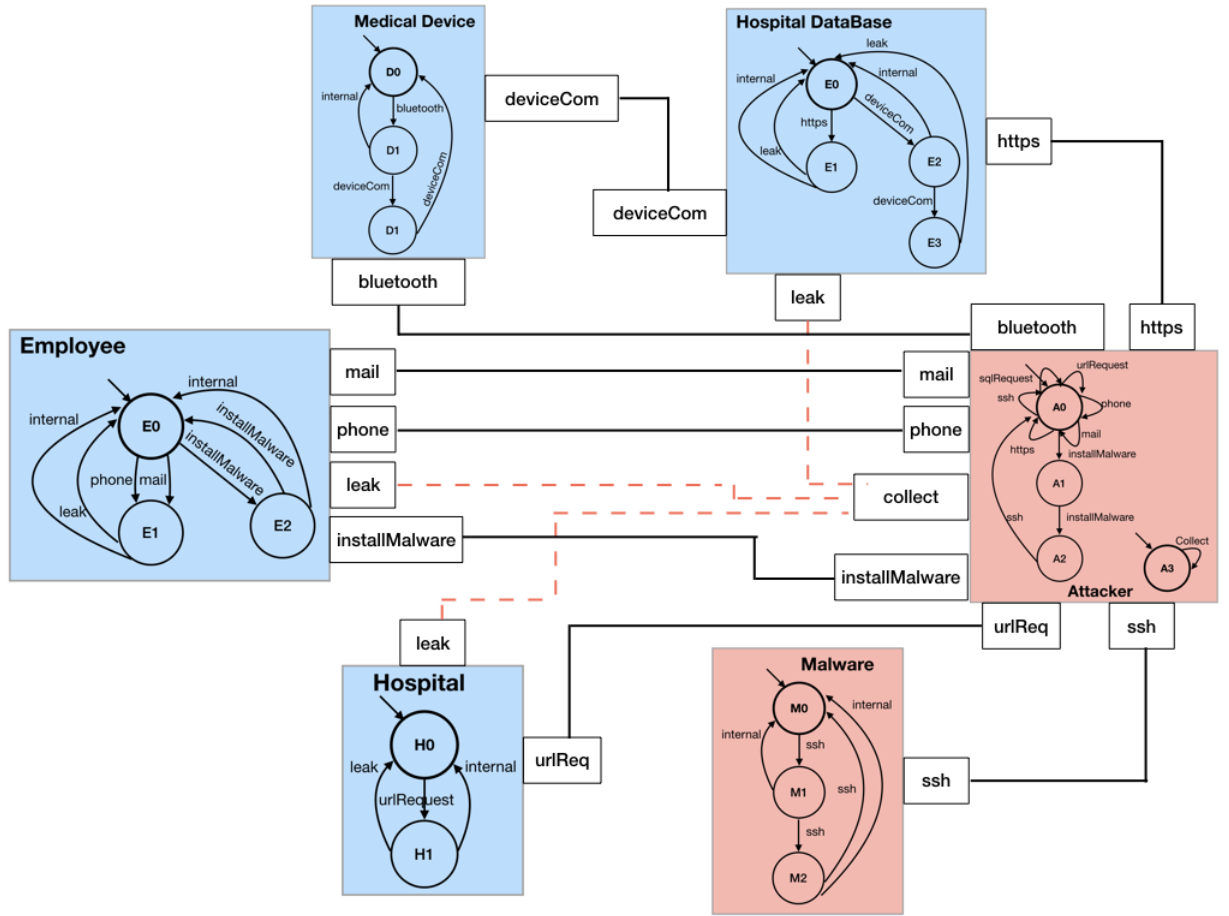


FIGURE 5.6: Smart Hospital BIP representation

Subsection 5.2.3. The two leaves "getCredentialPhishing" and "employee1email" represent values exchanged between the attacker and other entities through Send/Receive and Leak/Collect interactions. There are two external ports: one for the Send/Receive interactions, and another for the Leak/Collect ones. So whenever the attacker exchanges values with another entity, one of these two ports are triggered, depending on the kind of communication. The monitor compares the value exchanged with the ones on the attack tree and if they match, the boolean value that corresponds to the leaf is set to true. For instance, let's consider that the employee leaks the value "employee1email" to the attacker. The monitor "leak_collect" external port is triggered. In Figure 5.7, we can see that in the leak_collect transition, the monitor verifies what value was leaked to the attacker. Once it finds a match, the boolean variable "bool_leak_collect_employee1email" corresponding to the action is set to true and the score of the monitor is increased to the height of the leaf, here to "2". The internal node "Phishing" is transformed in BIP into an internal port as well as in a boolean variable. For the internal port to be triggered, both boolean variable corresponding to its leaves, as it is an "AND" type of node, must be true. It represents the fact that both steps of this subgoal are complete. If they are true, during the transition, the boolean variable corresponding to the internal node, here "bool_Phishing" is set to true and the score variable is incremented.

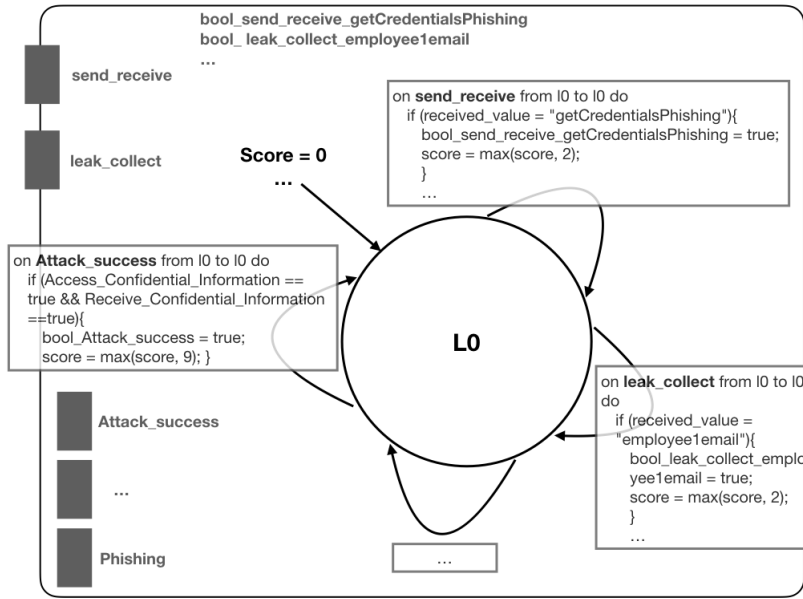


FIGURE 5.7: Smart Hospital BIP monitor

5.2.5 Experiments

In this part, we conducted the statistical analysis on the model that we introduced in Subsection 5.2.1. The results are presented in Figure 5.8. They are composed of three tables that display the probabilities of the success of an attack considering different probabilities in the model.

Table 1 In this table, there are no probabilities in the model which leads to a uniform distribution for all choices. This means that, when given a choice, an entity has the same probabilities to leak information than to ignore the malicious request. We can see here that the chances of success of the attacker cannot be considered to be a rare event. In this case, Monte Carlo is faster than Importance Splitting and gives precise results even with the lowest number of simulation.

As for Importance Splitting, it is precise and not significantly slower than Monte Carlo for 1 000 simulations. But we can see that from the highest number of simulations, Importance Splitting starts behaving incorrectly.

Table 2 In this example, we gave 50 more chances for an entity to ignore a malicious request from the attacker than to leak confidential data. To do so, we put a weight of "1" to all leak actions and a weight of "50" to their internal action counterparts. The result is that the possibilities of success of the attacker are fewer, without being a rare event.

Monte Carlo gives less specific results than in the first simulation, even with the highest number of simulations. Importance Splitting is twice slower than Monte Carlo but more specific for the first two simulations. However, for the highest number of simulations, it is giving incoherent results.

Table 3 This model decreases again the chances of success of the attacker. Here, the weight of the leak actions is 1 while the internal counterpart action has a weight of 100. Under 10000 simulation it seems impossible for the Monte

Carlo algorithm to give an estimate. With the two highest number of simulations, Monte Carlo can give us a value approach but there is nothing specific. The estimate is considerably smaller than the number of simulations, and thus we don't have a lot of confidence in Monte Carlo. Importance splitting helps here to confirm, or add confidence in the result. Alternatively, Monte Carlo for a large number of simulations, can confirm the results of importance splitting on a few simulations.

Table 1: No Weights

Smart Hospital: Model 1				
Nb of Simulations	Monte Carlo		Importance Splitting	
	Time (s)	Result	Time (s)	Result
1 000	6,9	1,244E-01	10,5	1,315E-01
10 000	52	1,195E-01	83,6	1,285E-01
100 000	506,9	1,221E-01	942,5	3,496E-02

Table 2: Weights leak=1; internal= 50

Smart Hospital: Model 50				
Nb of Simulations	Monte Carlo		Importance Splitting	
	Time (s)	Result	Time (s)	Result
1 000	7,4	1,0E-04	13	1,471E-04
10 000	55,5	1,0E-04	103,9	4,124E-05
100 000	541,7	8,0E-05	957,7	9,676E-10

Table 3: Weights leak=1; internal= 100

Smart Hospital: Model 100				
Nb of Simulations	Monte Carlo		Importance Splitting	
	Time (s)	Result	Time (s)	Result
1 000	12,4	0,0E+00	22,6	1,512E-05
10 000	95	1,0E-05	189,7	5,416E-06
100 000	554,2	1,0E-05	842	4,931E-09

FIGURE 5.8: Smart Hospital Experiments Results

5.3 Amazon Key

This section describes the Amazon Key use case. After a brief description of the system and its general concepts in Subsection 5.3.1, we introduce the corresponding SOML model in Subsection 5.3.2. Then we describe the possible and existing attacks towards the system that we included in the attack tree of the Subsection 5.3.3. The Subsection 5.3.4 presents the corresponding BIP model and Subsection 5.3.5 shows the results of the statistical analysis, performed on this use case.

5.3.1 Overview

Amazon Key is a service, provided by Amazon, that allows customers to have an automated control over the access to their house. Using a Smart Lock, the customer can allow other people to enter his home while he is not there. A set of cameras provides him with real time images of his house, supposed to increase the security of the delivery.

The door can be controlled by a code, sent to the Smart Lock, using a smart phone or another device. The code can be a permanent one, used by the people living in the house, or temporary. The temporary code is meant to be used by a specific and punctual visitor: a guest, a deliverer or a house worker for instance. In the case of a visit, the Amazon cloud cam gets involved for security reasons. It records the front door of the house owner, allowing him to have a real-time footage of the process. The Cloud Cam and the Smart Lock are connected via a low-power ZigBee connection. The camera is notified to start and stop recording by the Smart Lock through this connection.

For a better understanding of the procedure, we illustrate an indoor delivery process in Figure 5.9.

The indoor delivery is meant for a customer to have his order delivered inside of his

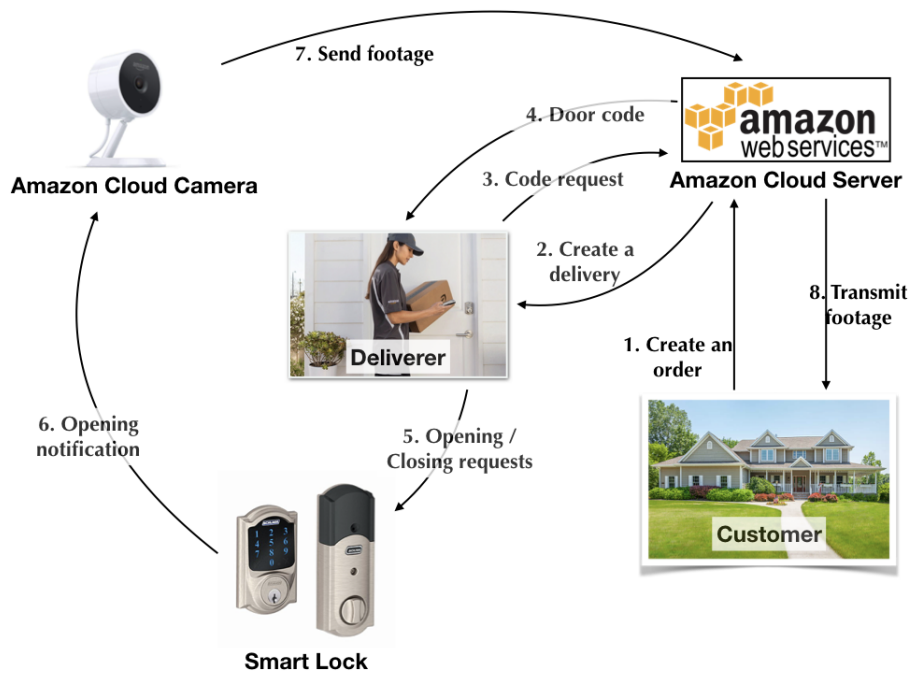


FIGURE 5.9: Amazon Key Behaviour

house while he is gone. The different steps of the procedure are showed in Figure 5.9 and are as follows:

1. The customer makes a purchase on the Amazon Website and asks for an indoor delivery. The customer must have an Amazon account and his house has to be equipped with the Smart Lock.
2. The Amazon server creates a delivery number, which figures on the package in the form of a barcode.
3. Once the package is ready, the deliverer goes to the drop point: the customer house. The deliverer then identifies himself by scanning the delivery barcode with an Amazon application. This application sends the order number to the Amazon Server. The Amazon Server is then notified that the delivery person needs to open the door.
4. If the delivery information provided are consistent, the Amazon Server replies with the temporary code for the Lock.
5. The deliverer sends this code to the Lock to unlock the door. If the code is accepted, the door is unlocked.
6. The Lock sends a notification to the camera via bluetooth or ZigBee protocol, to start recording.
7. The recording is sent to the Amazon Server.
8. The Amazon Server transmits the footage to the customer who can watch it in real time or access it later.
9. Once the package is dropped, the delivery person sends a lock instruction to the Smart Lock, which also notifies the camera to stop recording, and leaves.

5.3.2 Amazon Key IoT Model

To model the system using SOML, we use five internal entities that we can see in Figure 5.10:

Customer	He is the one requesting the delivery. His initial information are his customer number that he uses to log on the Amazon Server. He also possesses an email and an address.
Cloud Camera	It is in charge of surveillance. The footage recorded are key to ensure the safe and successful conduct of the delivery. It also has an identification number to be recognized by the Amazon Server and the Smart Lock.
Smart Lock	It can be opened via a code either temporary or permanent, depending on the needs. It communicates via bluetooth or ZigBee protocol with the camera, to notify it of when to start and stop recording.
Amazon Server	It is in charge of the coordination of the events. He makes the connection between the camera and the customer by transmitting the footage. He is also in charge of creating door codes upon request of the customer and sending them to different actors of the system.

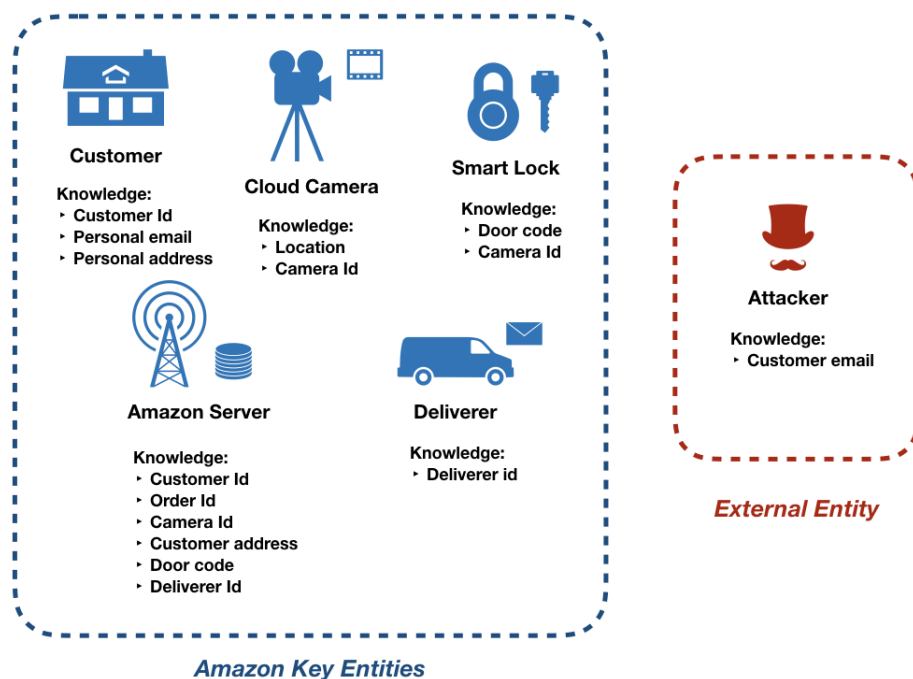


FIGURE 5.10: Initial State of the Amazon Key System

Deliverer	He is an Amazon employee and therefore has an id that allows him to communicate with the Server. This communication allows him to access his different tasks and receive the information necessary to accomplish them.
Attacker	He starts his attack with only the customer's email address. His final goal is to gather enough information in order to get inside of the house without being seen. This means that he has to open the door and disable the camera.

We defined several protocols for this model, each of them supported by a different verification.

Customer Communication	It models the exchanges between the Amazon Server and the customer that can happen at different stages of the delivery. The security of these interactions is based on the knowledge of the customer identification number, that is known only by the customer himself.
Staff Communication	This is used by the Amazon Server and the deliverer to communicate outside of the delivery process. This protocol is used for instance when the Amazon Server assigns a task to the deliverer. The verification is made on the deliverer identification number.
Delivery	This protocol is used for the communication between the deliverer and the Amazon Server during a delivery. The communication is allowed only if the deliverer sends the correct delivery number to the Amazon Server that answers by sending him a door code.

Door Control	The exchange between the Smart Lock and people who want to unlock the door. The verification is performed on the code sent to the Smart Lock. When trying to send an incorrect code to the lock, the communication is not possible.
Camera Detection	This represents the fact that to be able to communicate with the camera, it is necessary to be in its communication range. This verification is made on the address of the entities as we consider it represents the physical location of an entity.
Camera Communication	This protocol allows to control the camera in order to ask it to start and stop recording, as well as to retrieve the footage. For instance, this happens when the lock notifies it to start or stop recording and when the camera sends the footage to the Amazon Server, to transmit it to the customer. This protocol bases the verification on the identification number of the camera.
Mail	The customer can be contacted by email, under the condition of knowing his email address.

5.3.3 Amazon Key under Attacks

There are two known attacks towards the Amazon Key system. The first one was posted on Twitter by a Bay Area security researcher, aims to disrupt the communication between the Smart Lock and the device used to open and close it (for instance during a delivery). This is achieved by using a "break and enter dropbox" device within the communication range of the lock. Details of the device were not made public by the researcher, but it seems to be some kind of minicomputer. What happens is that once the delivery person arrives to the customer house, he opens the door using the code, drops the package. Once this is done, he hits the lock command on his smartphone and leaves. Because of the malicious device, the lock request doesn't work and the door remains open. The attacker can just open the door and walk inside the house once the deliverer left.

The second attack was discovered by Rhino Labs Security [2]. They found that by using known wifi vulnerabilities, they can disconnect the camera from its router.

In our simulation we consider an attack successful if the attacker manages to open the door and to disable the camera. This way he can enter the house without being spotted. The attack showed in the attack tree of the Figure 5.11 are inspired by the existing hacks. The house is considered vulnerable if the attacker manages to open the door and to disable the camera.

To open the door the attacker needs to know the house location. This can be achieved in two ways: either the customer leaks his identifier, which allows the attacker to connect to his profile and then get access to the information. The second way is that we consider that the customer's computer is infected. If this is the case, the attacker can access to the order validation of Amazon. We model this with a leak of the order information to the attacker from Amazon.

Once the attacker knows the location and hour of the delivery, he needs to be able to get the door code. Using the hack posted on Twitter, we consider that there is a chance for the attacker to intercept the code sent to the Smart Lock by the deliverer

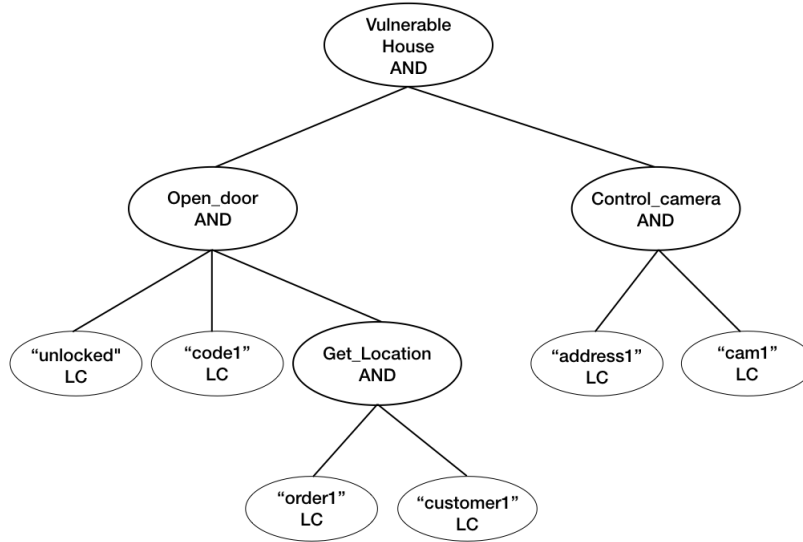


FIGURE 5.11: Amazon Key Attack Tree

when he opens the door. We model that with a possible leak action from the deliverer towards the attacker. If the attacker has access to the door code, he has to send a request to the Smart Lock through the door control protocol and the Smart Lock can respond positively or not to this code. We model the attacker successfully opening the door with a leak of the value "unlocked" from the Smart Lock to the Attacker, otherwise the Smart Lock will just ignore the attacker's request.

Adding more steps to the door opening process that there are in the real hack is meant to add difficulty for the attacker and then model the uncertainty that exists in real life when hacking a system.

The other requirement for the attack to succeed is that the camera is disabled. To do so, first the attacker needs to be physically in the communication range of the camera. We model this using the "camera detection" protocol. To be able to detect the camera and so to communicate with it, the attacker must know the address of the customer. This is made possible through another phishing email that asks the customer for this information. The customer can leak his address or just ignore the email. Once the attacker is able to detect the camera, he will try to hack it. This is modeled by a request made by the attacker: the camera will, or not, leak its identifier. If the answer of the camera is positive and the attacker possesses the camera identifier in his knowledge, we consider that he can control the camera and therefore, the second requirement of the attack is complete.

5.3.4 Amazon Key BIP Model

The transformation towards a BIP model using the SOML model of the Amazon Key system and the attack tree follows the same principles that we detailed previously, in Subsection 5.2.4.

Figure 5.12 shows part of the BIP model. For readability reason, the component variables and internal ports are not shown and the exported ports are limited to one for each protocol used by the entity.

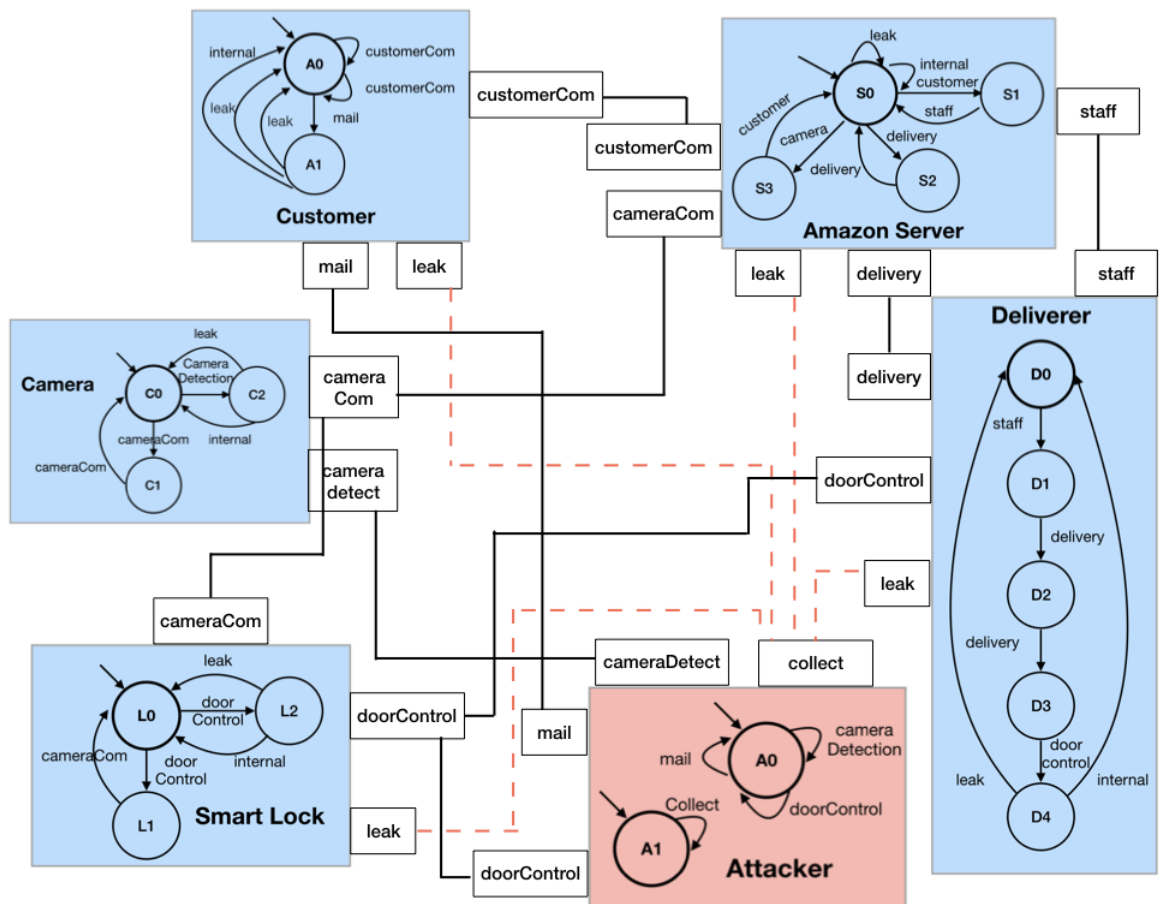


FIGURE 5.12: Amazon Key BIP representation

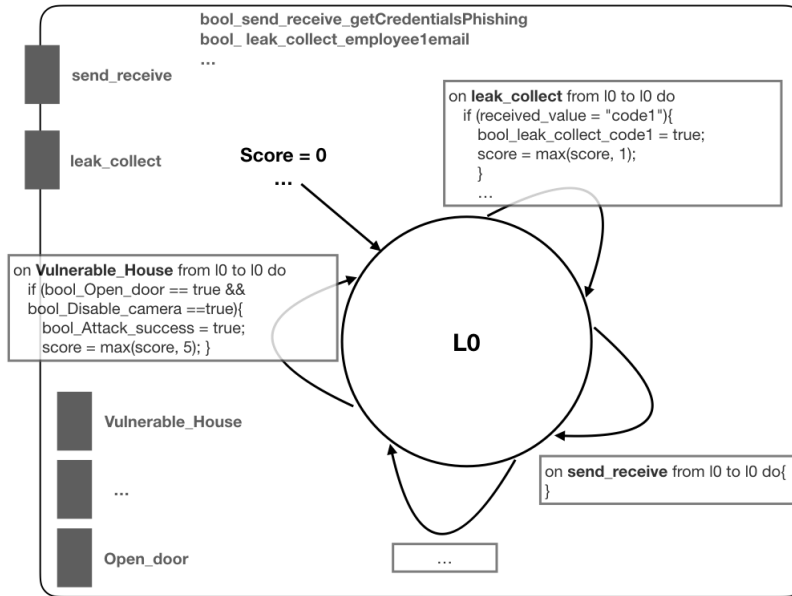


FIGURE 5.13: Amazon Key BIP monitor

Figure 5.13 shows part of the monitor for readability reasons. For this model, the attack steps are only composed of information leaks. No Send/Receive interaction is considered. As we can see in the monitor representation, the "send_receive" port doesn't do anything. The value comparisons lead to the same update of the boolean variables and of the score variable as seen before.

5.3.5 Experiments

In this subsection, we present the results of the experiments that we conducted on the Amazon Key model. The results are presented in Figure 5.14. As previously the probabilities of the model vary as we want to make the success of the attack a rare event. The approach is the same as for the previous model, Table 1 shows the results of the experiments performed on a model without any probabilities. We slowly decrease the chances of success, until the completion of the attack becomes a rare event.

More experiments have been performed in this case than previously because the balance between the attack success becoming a rare event and the memory resources for the experiment becoming insufficient was harder to find.

Table 1

Here, the attack success is not a rare event and both algorithms are equivalent in terms of accuracy as we can see that they converge towards the same value. The importance splitting algorithm takes more time but the time difference is not significant. We can see that the chances of success of the attack are lower than in the previous use case even though the attack tree is smaller and seems easier to perform. This can be explained by the fact that the previous attack tree included more paths and alternatives for the attack success and required for the attacker to get access to less distinct information.

Table 2	Even though in this second model the attack success is not a rare event yet, we can see that the Monte Carlo algorithm becomes less specific.
Table 3 and Table 4	To give an estimate of the success probabilities, Monte Carlo needs at least 10000 simulations.
Table 5	In this last simulation, the attack success is a rare event. We can see that with the available resources to perform the experiments, it seems impossible to have an accurate result using the Monte Carlo algorithm. However, with the use of the same resources and number of simulations, Importance Splitting is able to compute results that seem accurate.

Table 1: No Weight

Amazon key: Model 1				
Nb of Simulations	Monte Carlo		Importance Splitting	
	Time (s)	Result	Time (s)	Result
1 000	8	5,29E-02	9,5	6,227E-02
10 000	58,8	5,246E-02	74,8	5,855E-02
100 000	568,8	5,191E-02	693,2	5,681E-02

Table 2: Weights leak=1; internal= 5

Amazon key: Model 5				
Nb of Simulations	Monte Carlo		Importance Splitting	
	Time (s)	Result	Time (s)	Result
1 000	6,7	1,200E-03	9,3	8,463E-04
10 000	48,8	7,8E-04	72,1	1,027E-03
100 000	515,4	9,0E-04	739,1	3,925E-04

Table 3: Weights leak=1; internal= 10

Amazon key: Model 10				
Nb of Simulations	Monte Carlo		Importance Splitting	
	Time (s)	Result	Time (s)	Result
1 000	6,5	0E+00	9	9,261E-05
10 000	47,7	4,0E-05	71,7	4,784E-05
100 000	464,2	5,6E-05	670,9	1,881E-05

Table 4: Weights leak=1; internal= 15

Amazon key: Model 15				
Nb of Simulations	Monte Carlo		Importance Splitting	
	Time (s)	Result	Time (s)	Result
1 000	7	0E+00	10,3	1,739E-04
10 000	50,6	4E-05	79,8	5,034E-05
100 000	498,8	5,0E-05	752,4	2,326E-05

Table 5: Weights leak=1; internal= 20

Amazon key: Model 20				
Nb of Simulations	Monte Carlo		Importance Splitting	
	Time (s)	Result	Time (s)	Result
1 000	6,5	0,0E+00	10,0	2,428E-05
10 000	44,8	0,0E+00	75,4	6,015E-06
100 000	469,2	1,0E-06	843,5	3,188E-06

FIGURE 5.14: Amazon Key Experiments Results

5.4 Experiments Conclusion

As we saw in this chapter, we conducted the experiments on two distinct IoT systems. First, the Smart Hospital shows how an important and complex system operates whereas the Amazon Key gave an overview of how IoT systems can be integrated, on a small scale, to the life of every individual. The size also makes a difference on the attack surface. This is visible for instance on the size of the attack tree: the Smart Hospital has a bigger attack tree, and we have seen that attacks have higher probabilities to succeed on it than on the Smart Lock. This can be explained by the fact that the Smart Hospital is composed of a large number of complex systems and this increases its vulnerability, as there are more ways to attack it and then to access its confidential information. On the other hand, the Smart Key is smaller and therefore more difficult to attack. Its attack tree is smaller as the attacker has to exploit very specific vulnerabilities in order to conduct the attack, and for most of the steps, there are no alternatives. Another element is that the hospital is a public system and there are more available public information, such as email addresses, its physical address, ... whereas the Smart Lock belongs to an actual person whose information are not as easy to access (even though with the increasing use of social media, individuals make their information more and more easy to obtain).

Concerning the Monte Carlo and the importance splitting algorithms that we used to compute the probabilities, our findings are these. When probabilities are high, Monte Carlo turns out to be faster than importance splitting for a same degree of accuracy. But when the probabilities get close to be rare events, Monte Carlo needs a higher number of simulations than importance splitting in order to compute a result, that, in the end, seems less relevant. Importance splitting can be considered to be more accurate when the number of simulations are low, but on a high number of simulations it behaves incorrectly and therefore cannot be trusted. We argue that both algorithms must be used in a complementary fashion in order to give accurate results.

The use of these two algorithms and the results that they provide validate the fact that our methodology allows us to calculate the chances of an attack whether it is a rare event or not.

Chapter 6

Conclusion

We started this work with the objective of computing the success probabilities of an attack on a given system. The attacks were performed by one or several attacker with the objective of accessing sensitive and confidential data. We started by finding a way to model an IoT system. We didn't find any formal language that was adapted to connected environments and therefore to IoT systems, and that allowed us to use probabilities. We then decided to implement our own language: Security Oriented Modeling Language (SOML).

SOML is presented in [4] that presents both its abstract and concrete semantics, both of them are expressed formally. SOML is meant to give users a way to describe an IoT system easily by expressing each actor and connected device as an entity. For all entities of the system, we define a knowledge and a behaviour. Then, by showing how an entity can expand its knowledge and access information that it shouldn't have we are able to express attacks happening within the system. SOML allows us to add probabilities on the actions of the entities. By varying the probabilities on their behaviour, we can make "illegal" actions more or less likely to happen and therefore have an impact on the chances for the attack to succeed.

This brings us to the second part of the process: once we have a system model with defined probabilities on the actions, we want to be able to run the experiments and to compute the actual attack probabilities. For that we need two additional steps: first, explicitly model the attack that we want to measure the chances of and run simulations of the system. The overall process was published in [5].

The simulations are run using the BIP language that provides us a way to make the entities interact and run the attack within the system. To be able to use BIP, we implemented a transpiler that transforms a SOML model into an equivalent BIP model. Using the formal expression of both languages, we proved this equivalence by showing a bisimulation between the languages in Section 4.3.3.

The attack is expressed within the model and through the behaviour of its entities, but we have no way to flag the success or failure of an attack. Indeed, during the simulations we see the entities interact with each other and get access to information but we have no specific idea of if this entity is supposed to get these information or not. To remedy this situation, we implement an external representation of the attacks using an attack tree. In the tree, we describe the attack by showing what information the attacker will try to access, and the types of communication he will use to do so. During the conversion process, the attack tree is transformed into a BIP component, called monitor, that interacts with the BIP system that corresponds to the system representation. The monitor's role is to observe the system's running, without interfering, and flag an attack success. To do so, he receives information from the system about the entities' actions and is able to know at a given time where is the attack at. Thanks to the monitor's following of the attack progress, we are able to determine when an attack has been conducted and compute its probabilities.

The probability part is carried out by Plasma lab, which is a statistical model checker. As we said earlier, we make the chances of success vary by modifying the probabilities within the system. When the attack has a probability to happen that is inferior to 10^{-6} , we call it a "rare event". Plasma lab is able to calculate the chances of success of an attack whether it is a rare event or not, using several algorithms that we describe in Section 5.1. We decided to use two different algorithms: Monte Carlo and Importance Splitting, that is specifically designed for rare events. Throughout the experiments we were able to compare the efficiency and accuracy of both algorithms. We conclude that both algorithms should be used in a complementary way. Indeed, when the probabilities are high, Monte Carlo tends to be more efficient, for a shorter time of simulations but requires a higher number of simulation as the probabilities decrease to keep on being accurate. When we arrive to a point of rare event, Monte Carlo requires a number of simulations that is too high for the resources we had to give us a result. On the other hand, Importance Splitting takes a longer time than Monte Carlo to compute probabilities when they are high. But as we get closer to a rare event, Importance Splitting is able to give us a more accurate result than Monte Carlo for the same number of simulations. For these reasons we argue that both algorithms have a complementary role towards each other. The simulations were applied on two different IoT models: a smart hospital and the Amazon Key. The models are very distinct in sizes and applications from each other. These use cases showed us that SOML and the subsequent probabilities analysis can be applied on a variety of IoT systems.

Throughout this work we created a new way of modeling connected environments using SOML. SOML is a formal language that puts an emphasis on how the different parts of the system communicate by showing the communication protocols and defining types of data. We also implemented a compiler that takes as inputs the SOML model and an attack tree and generate an executable model that allows us to perform simulations of attacks towards the model. As we separate the system model and the attack expression, it is possible to play various kind of assaults on one model. Then, by coupling the engine that plays the simulations with a statistical model checker we calculate the probabilities for the attack to succeed.

One of the limitation of this work is the manual implementation of the attack and of the attacker. An improvement would be to have a visual representation of the language that would enable users to create the representation of the system via a visual interface. The SOML code would be automatically generated from this graphical view. Also, to implement the attack tree, we had to look for real life attacks that happened on similar system. We then had to create one or several attack entities and design their behaviour so it would match the attack tree. An improvement would be to be able to automatically generate attackers and attack trees related to a system. A way of doing it would be to label some parts of the system as critical and tag some of the system vulnerabilities. From there, we could generate a set of attacks which goal is to access the confidential parts of the system. An attacker would also be generated as an entity who exploits the vulnerabilities of the system in order to conduct an attack.

Bibliography

- [1] A. B. Abkenar, S. W. Loke, and A. Zaslavsky. “IoT-Enabled Group Activity Recognition Services Using a Modeling Language Approach”. In: *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*. 2018, pp. 1–6. DOI: [10.1109/IoT-SIU.2018.8519854](https://doi.org/10.1109/IoT-SIU.2018.8519854).
- [2] *Amazon Key Security: CloudCam Subject to Disruption Attacks*. 2018. URL: <https://rhinosecuritylabs.com/internet-of-things/amazon-key-security-cloudcam-disruption-attacks/>.
- [3] Luigi Atzori, Antonio Iera, and Giacomo Morabito. “The Internet of Things: A Survey”. In: *Comput. Netw.* 54.15 (Oct. 2010), pp. 2787–2805. ISSN: 1389-1286. DOI: [10.1016/j.comnet.2010.05.010](https://doi.org/10.1016/j.comnet.2010.05.010). URL: <http://dx.doi.org/10.1016/j.comnet.2010.05.010>.
- [4] D. Beaulaton et al. “A Language for Analyzing Security of IOT Systems”. In: *2018 13th Annual Conference on System of Systems Engineering (SoSE)*. 2018, pp. 37–44. DOI: [10.1109/SYSOSE.2018.8428704](https://doi.org/10.1109/SYSOSE.2018.8428704).
- [5] D. Beaulaton et al. “Security Analysis of IoT Systems using Attack Trees”. In: *GramSec 2019*. 2019.
- [6] T. Ben Hassine, O. Khayati, and H. Ben Ghezala. “An IoT domain meta-model and an approach to software development of IoT solutions”. In: *2017 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC)*. 2017, pp. 32–37. DOI: [10.1109/IINTEC.2017.8325909](https://doi.org/10.1109/IINTEC.2017.8325909).
- [7] Saddek Bensalem et al. “Statistical Model Checking Qos Properties of Systems with SBIP”. In: *Proceedings of the 5th International Conference on Leveraging Applications of Formal Methods, Verification and Validation: Technologies for Mastering Change - Volume Part I*. 2012. DOI: [10.1007/978-3-642-34026-0_25](https://doi.org/10.1007/978-3-642-34026-0_25).
- [8] S. Bistarelli, F. Fioravanti, and P. Peretti. “Defense trees for economic evaluation of security investments”. In: *First International Conference on Availability, Reliability and Security (ARES'06)*. 2006, 8 pp.–423. DOI: [10.1109/ARES.2006.46](https://doi.org/10.1109/ARES.2006.46).
- [9] Stefano Bistarelli, Marco Dall’Aglio, and Pamela Peretti. “Strategic Games on Defense Trees”. In: vol. 4691. Sept. 2007, pp. 1–15. DOI: [10.1007/978-3-540-75227-1_1](https://doi.org/10.1007/978-3-540-75227-1_1).
- [10] Benoit Boyer et al. “PLASMA-lab: A Flexible, Distributable Statistical Model Checking Library”. In: vol. 8054. Aug. 2013, pp. 160–164. DOI: [10.1007/978-3-642-40196-1_12](https://doi.org/10.1007/978-3-642-40196-1_12).
- [11] Bryant. “Graph-Based Algorithms for Boolean Function Manipulation”. In: *IEEE Transactions on Computers* C-35.8 (1986), pp. 677–691. ISSN: 0018-9340. DOI: [10.1109/TC.1986.1676819](https://doi.org/10.1109/TC.1986.1676819).

- [12] J. R. Burch et al. “Sequential Circuit Verification Using Symbolic Model Checking”. In: *Proceedings of the 27th ACM/IEEE Design Automation Conference*. DAC '90. Orlando, Florida, USA: ACM, 1990, pp. 46–51. ISBN: 0-89791-363-9. DOI: [10.1145/123186.123223](https://doi.org/10.1145/123186.123223). URL: <http://doi.acm.org/10.1145/123186.123223>.
- [13] D. R. Cacciagrano and R. Culmone. “Formal Semantics of an IoT-Specific Language”. In: *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. 2018, pp. 579–584. DOI: [10.1109/WAINA.2018.00148](https://doi.org/10.1109/WAINA.2018.00148).
- [14] Sergio Caltagirone, Andy Pendergast, and Chris Betz. “The Diamond Model of Intrusion Analysis A Summary By Sergio Caltagirone”. In: 2013.
- [15] Edmund M. Clarke. “Model checking”. In: *Foundations of Software Technology and Theoretical Computer Science*. Ed. by S. Ramesh and G. Sivakumar. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 54–56. ISBN: 978-3-540-69659-9.
- [16] Edmund M. Clarke and E. Allen Emerson. “DESIGN AND SYNTHESIS OF SYNCHRONIZATION SKELETONS USING BRANCHING TIME TEMPORAL LOGIC”. In: *25 Years of Model Checking: History, Achievements, Perspectives*. Ed. by Orna Grumberg and Helmut Veith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 196–215. ISBN: 978-3-540-69850-0. DOI: [10.1007/978-3-540-69850-0_12](https://doi.org/10.1007/978-3-540-69850-0_12). URL: https://doi.org/10.1007/978-3-540-69850-0_12.
- [17] Bruno Costa, Paulo F. Pires, and Flavia C. Delicato. “Modeling SOA-Based IoT Applications with SoaML4IoT”. In: *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. 2019.
- [18] F. Cuppens and A. Mieke. “Alert correlation in a cooperative intrusion detection framework”. In: *Proceedings 2002 IEEE Symposium on Security and Privacy*. 2002, pp. 202–215. DOI: [10.1109/SECPRI.2002.1004372](https://doi.org/10.1109/SECPRI.2002.1004372).
- [19] Marc Dacier and Yves Deswarte. “Privilege graph: An extension to the typed access matrix model”. In: *Computer Security — ESORICS 94*. Ed. by Dieter Gollmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 319–334. ISBN: 978-3-540-49034-0.
- [20] K. S. Edge et al. “Using Attack and Protection Trees to Analyze Threats and Defenses to Homeland Security”. In: *MILCOM 2006 - 2006 IEEE Military Communications conference*. 2006, pp. 1–7. DOI: [10.1109/MILCOM.2006.302512](https://doi.org/10.1109/MILCOM.2006.302512).
- [21] ENISA. *Smart Hospitals, Security and Resilience for Smart Health Service and Infrastructures*. Tech. rep. 2016.
- [22] T. Eterovic et al. “An Internet of Things visual domain specific modeling language based on UML”. In: *2015 XXV International Conference on Information, Communication and Automation Technologies (ICAT)*. 2015, pp. 1–5. DOI: [10.1109/ICAT.2015.7340537](https://doi.org/10.1109/ICAT.2015.7340537).
- [23] O. Flaten and Mass Soldal Lund. “How Good are Attack Trees for Modelling Advanced Cyber Threats”. In: 2014.
- [24] B. Foo et al. “ADEPTS: adaptive intrusion response using attack graphs in an e-commerce environment”. In: *2005 International Conference on Dependable Systems and Networks (DSN'05)*. 2005, pp. 508–517. DOI: [10.1109/DSN.2005.17](https://doi.org/10.1109/DSN.2005.17).

- [25] Rob Van Glabbeek, Scott Smolka, and Bernhard Steffen. “Reactive, Generative, and Stratified Models of Probabilistic Processes”. In: *Information and Computation* 121.1 (1995). DOI: <https://doi.org/10.1006/inco.1995.1123>.
- [26] Imene Ben Hafaiedh, Susanne Graf, and Sophie Quinton. “Building Distributed Controllers for Systems with Priorities”. In: *J. Log. Algebr. Program.* 80.3 (2011), pp. 194–218.
- [27] K. Havelund et al. *Formal Methods: 22nd International Symposium, FM 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 15–17, 2018, Proceedings*. Lecture Notes in Computer Science. Springer International Publishing, 2018. ISBN: 9783319955827. URL: <https://books.google.fr/books?id=LtNjDwAAQBAJ>.
- [28] Mats Heimdahl and N.G. Leveson. “Completeness and Consistency Analysis of State-Based Requirements”. In: Jan. 1995, pp. 3–3. ISBN: 0-89791-708-1.
- [29] Guy Helmer et al. “A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System”. In: *Requirements Engineering* 7.4 (2002), pp. 207–220. ISSN: 1432-010X. DOI: [10.1007/s007660200016](https://doi.org/10.1007/s007660200016). URL: <https://doi.org/10.1007/s007660200016>.
- [30] Guy Helmer et al. “Software Fault Tree and Colored Petri Net Based Specification, Design and Implementation of Agent-Based Intrusion Detection Systems”. In: *International Journal of Information and Computer Security* 1 (Sept. 2002).
- [31] Amin Hosseinian-Far, Hamid Jahankhani, and Elias Pimenidis. “Evaluating Influence Diagrams”. In: Mar. 2012.
- [32] Cyrille Jegourel, Axel Legay, and Sean Sedwards. “A Platform for High Performance Statistical Model Checking – PLASMA”. In: Mar. 2012, pp. 498–503. DOI: [10.1007/978-3-642-28756-5_37](https://doi.org/10.1007/978-3-642-28756-5_37).
- [33] Cyrille Jegourel, Axel Legay, and Sean Sedwards. “Importance Splitting for Statistical Model Checking Rare Properties”. In: *Computer Aided Verification*. Ed. by Natasha Sharygina and Helmut Veith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 576–591. ISBN: 978-3-642-39799-8.
- [34] Qi Jing et al. “Security of the Internet of Things: perspectives and challenges”. In: *Wireless Networks* 20.8 (2014), pp. 2481–2501. ISSN: 1572-8196. DOI: [10.1007/s11276-014-0761-7](https://doi.org/10.1007/s11276-014-0761-7). URL: <https://doi.org/10.1007/s11276-014-0761-7>.
- [35] C. B. Jones. “Theorem proving and software engineering”. In: *Software Engineering Journal* 3.1 (1988), pp. 2–. DOI: [10.1049/sej.1988.0001](https://doi.org/10.1049/sej.1988.0001).
- [36] H. Kahn and Harris. “Estimation of particle transmission by random sampling”. In: *Bureau of Standards applied mathematics series* (1951).
- [37] Barbara Kordy, Marc Pouly, and Patrick Schweitzer. “Computational Aspects of Attack–Defense Trees”. In: *Security and Intelligent Information Systems*. Ed. by Pascal Bouvry et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 103–116.
- [38] Barbara Kordy et al. “Foundations of Attack–Defense Trees”. In: *Formal Aspects of Security and Trust*. Ed. by Pierpaolo Degano, Sandro Etalle, and Joshua Guttman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 80–95. ISBN: 978-3-642-19751-2.

- [39] M. Kwiatkowska, G. Norman, and D. Parker. “PRISM 4.0: Verification of Probabilistic Real-time Systems”. In: *Proc. 23rd International Conference on Computer Aided Verification (CAV’11)*. Ed. by G. Gopalakrishnan and S. Qadeer. Vol. 6806. LNCS. Springer, 2011, pp. 585–591.
- [40] TrapX Labs. *Anatomy of an Attack MEDJACK (Medical Device Hijack)*. Tech. rep. 2015.
- [41] TrapX Labs. *MEDJACK 2.0 Hospitals under siege*. Tech. rep. 2016.
- [42] H. S. Lallie, K. Debattista, and J. Bal. “An Empirical Evaluation of the Effectiveness of Attack Graphs and Fault Trees in Cyber-Attack Perception”. In: *IEEE Transactions on Information Forensics and Security* 13.5 (2018), pp. 1110–1122. ISSN: 1556-6013. DOI: [10.1109/TIFS.2017.2771238](https://doi.org/10.1109/TIFS.2017.2771238).
- [43] Kim G. Larsen et al. “Scenario-based Analysis and Synthesis of Real-time Systems Using UPPAAL”. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. DATE ’10. Dresden, Germany: European Design and Automation Association, 2010, pp. 447–452. ISBN: 978-3-9810801-6-2. URL: <http://dl.acm.org/citation.cfm?id=1870926.1871032>.
- [44] Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. “Plasma Lab: A Modular Statistical Model Checking Platform”. In: vol. 9952. Oct. 2016, pp. 77–93. DOI: [10.1007/978-3-319-47166-2_6](https://doi.org/10.1007/978-3-319-47166-2_6).
- [45] Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. “Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains”. In: *Leading Issues in Information Warfare and Security Research* 1 (Jan. 2011).
- [46] A. Marback et al. “Security test generation using threat trees”. In: *2009 ICSE Workshop on Automation of Software Test*. 2009, pp. 62–69. DOI: [10.1109/IWAST.2009.5069042](https://doi.org/10.1109/IWAST.2009.5069042).
- [47] Sjouke Mauw and Martijn Oostdijk. “Foundations of Attack Trees”. In: *Information Security and Cryptology - ICISC 2005*. Ed. by Dong Ho Won and Seungjoo Kim. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 186–198. ISBN: 978-3-540-33355-5.
- [48] J. P. McDermott. “Attack Net Penetration Testing”. In: *Proceedings of the 2000 Workshop on New Security Paradigms*. NSPW ’00. Ballycotton, County Cork, Ireland: ACM, 2000, pp. 15–21. ISBN: 1-58113-260-3. DOI: [10.1145/366173.366183](https://doi.org/10.1145/366173.366183). URL: <http://doi.acm.org/10.1145/366173.366183>.
- [49] R. Milner. *A Calculus of Communicating Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1982. ISBN: 0387102353.
- [50] D. P. Mirembe and M. Muyeba. “Threat Modeling Revisited: Improving Expressiveness of Attack”. In: *2008 Second UKSIM European Symposium on Computer Modeling and Simulation*. 2008, pp. 93–98. DOI: [10.1109/EMS.2008.83](https://doi.org/10.1109/EMS.2008.83).
- [51] Yisroel Mirsky et al. “CT-GAN: Malicious Tampering of 3D Medical Imagery using Deep Learning”. In: *CoRR* abs/1901.03597 (2019). arXiv: [1901.03597](https://arxiv.org/abs/1901.03597). URL: <http://arxiv.org/abs/1901.03597>.
- [52] I. Morikawa and Y. Yamaoka. “Threat Tree Templates to Ease Difficulties in Threat Modeling”. In: *2011 14th International Conference on Network-Based Information Systems*. 2011, pp. 673–678. DOI: [10.1109/NBiS.2011.113](https://doi.org/10.1109/NBiS.2011.113).

- [53] Peng Ning, Yun Cui, and Douglas S. Reeves. “Analyzing Intensive Intrusion Alerts via Correlation”. In: *Recent Advances in Intrusion Detection*. Ed. by Andreas Wespi, Giovanni Vigna, and Luca Deri. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 74–94. ISBN: 978-3-540-36084-1.
- [54] S. Noel et al. “Efficient minimum-cost network hardening via exploit dependency graphs”. In: *19th Annual Computer Security Applications Conference, 2003. Proceedings.* 2003, pp. 86–95. DOI: [10.1109/CSAC.2003.1254313](https://doi.org/10.1109/CSAC.2003.1254313).
- [55] P. Ongsakorn et al. “Cyber threat trees for large system threat cataloging and analysis”. In: *2010 IEEE International Systems Conference*. 2010, pp. 610–615. DOI: [10.1109/SYSTEMS.2010.5482351](https://doi.org/10.1109/SYSTEMS.2010.5482351).
- [56] Terence Parr. Ed. by Susannah Davidson Pfalzer. *The Pragmatic Programmers*. ISBN: 1934356999.
- [57] C. A. Petri. “Communication with automata”. In: 1966.
- [58] D. A. Robles-Ramirez, P. J. Escamilla-Ambrosio, and T. Tryfonas. “IoTsec: UML Extension for Internet of Things Systems Security Modelling”. In: *2017 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE)*. 2017, pp. 151–156. DOI: [10.1109/ICMEAE.2017.20](https://doi.org/10.1109/ICMEAE.2017.20).
- [59] A. Rodriguez-Mota et al. “Towards IoT cybersecurity modeling: From malware analysis data to IoT system representation”. In: *2016 8th IEEE Latin-American Conference on Communications (LATINCOM)*. 2016, pp. 1–6. DOI: [10.1109/LATINCOM.2016.7811597](https://doi.org/10.1109/LATINCOM.2016.7811597).
- [60] Enno Ruijters and Mariëlle Stoelinga. “Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools”. In: *Computer Science Review* 15-16 (2015), pp. 29–62. ISSN: 1574-0137. DOI: <https://doi.org/10.1016/j.cosrev.2015.03.001>. URL: <http://www.sciencedirect.com/science/article/pii/S1574013715000027>.
- [61] Enno Ruijters et al. “Rare Event Simulation for Dynamic Fault Trees”. In: *Computer Safety, Reliability, and Security*. Ed. by Stefano Tonetta, Erwin Schoitsch, and Friedemann Bitsch. Cham: Springer International Publishing, 2017, pp. 20–35. ISBN: 978-3-319-66266-4.
- [62] Bruce Schneier. “Attack Trees”. In: *Dr. Dobbs’s Journal* 24.12 (1999), pp. 21–29.
- [63] Edel Sherratt et al. “SDL - The IoT Language”. In: *SDL 2015: Model-Driven Engineering for Smart Cities*. Ed. by Joachim Fischer et al. Cham: Springer International Publishing, 2015, pp. 27–41. ISBN: 978-3-319-24912-4.
- [64] Jan Steffan and Markus Schumacher. “Collaborative Attack Modeling”. In: *Proceedings of the 2002 ACM Symposium on Applied Computing*. SAC ’02. Madrid, Spain: ACM, 2002, pp. 253–259. ISBN: 1-58113-445-2. DOI: [10.1145/508791.508843](https://doi.org/10.1145/508791.508843). URL: <http://doi.acm.org/10.1145/508791.508843>.
- [65] Terry Tidwell et al. “Modeling Internet Attacks”. In:
- [66] Stilianos Vidalis and andH. Lewis Jones. “Using Vulnerability Trees for Decision Making in Threat Assessment”. In: 2003.
- [67] H.A. Watson. In: *Launch control safety study* 1.24 (1961).