Doctoral Dissertation

Automated Secure Design of Networked Systems

Ooi Sian En

Supervisor: Assoc. Prof. Razvan Beuran

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Information Science)

March 2023

Abstract

The trend of Digital Transformation (DX) to modernize the society introduces various ICT challenges. DX ICT often requires a high frequency of change and emphasises speed, flexibility and efficiency, which as a result, requires an agile system delivery method. Conventionally, systems are manually designed based on tacit and explicit knowledge of the system designer. While manual design may work for small or relatively simple systems, the tractability of a system is quickly lost for systems of systems, which are common in Society 5.0. The challenges are even greater when considering securing the system, especially when frequent changes require the reassessment of the system's security to ensure that those changes did not degrade the security of the intended system.

This research aims to improve the security of a system by introducing an automated verification of the security characteristics of a system into fundamental system design. An automated secure design framework, SecureWeaver, was proposed and implemented, which consists of contributions such as a knowledge base for secure design and security verification algorithms to verify the generated design. In addition, case studies on IoT applications, such as end-to-end communication and secure configuration implementation were carried out. The results of the case studies were also used as the motivating evaluations for SecureWeaver. A set of models for IT/NW and IoT system design were developed to evaluate SecureWeaver, which cover scenarios such as typical corporate network, IoT appliances, and also hardware-level system design in an automated and secure manner.

The evaluation showed that SecureWeaver is able to generate a system design that mitigates the security threats present in the input requirements via the automatic placement of security-based components in the system design. The performance characteristics of SecureWeaver also demonstrated that the security verification overhead compared to the total system design time is largest for simple scenarios, for which the actual design is very fast, still being just 0.58% in such a case. The expected impact of this dissertation is to decrease the human effort in system design via automatically designing secure systems by using the proposed framework. This formalized design approach will also add to the knowledge field in automatic system design.

Keywords: networked systems, secure system design, automated design, design space exploration, MITRE ATT&CK.

Acknowledgment

I would like to express my deepest gratitude and appreciation to my supervisor, Associate Professor Razvan Beuran, for his continuous guidance, patience and supervision throughout my doctoral journey. His advice, encouragement, motivation, suggestion and valuable insights provided me the means to complete this dissertation. I would also like to express my gratitude to my co-supervisor, Professor Yasuo Tan for his support throughout my entire postgraduate studies at JAIST.

I also wish to express my gratitude to my minor research supervisor, Associate Professor Yuto Lim for his guidance and support. I would also like to express my gratitude to Professor Yoichi Shinoda for his insightful comments on the dissertation. Furthermore, I wish to express my sincere thanks to the researchers from the Secure System Platform Research Laboratories at NEC Corporation, Dr. Takayuki Kuroda, Mr. Takuya Kuwahara, Dr. Ryosuke Hotchi, Mr. Norihito Fujita for imparting their extensive knowledge in the field of automated system designer.

I would like to thank my dear wife for her unwavering patience and support throughout my postgraduate journey. Last but not least, I would like to thank my family for their support, in which they always provided me with more than enough for me to explore and turn ideas and dreams into realities.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Digital transformation efforts in recent years, some of them related to the COVID-19 pandemic, have made the network infrastructure a core element of our society that enables virtual communication, e-commerce activities, telecommuting, and so on. The current model for designing and deploying networked systems is that the customers express their needs in service-level requirement descriptions, and system developers focus mainly on resource-level requirements [7]. However, there is no direct way to "translate" between these two types of requirements, and expert knowledge must be used to determine the system components to be deployed and the manned in which they should be integrated in order to meet customer needs.

In order to formalize the expression of network service requirements, several classed of approaches have been proposed, such as intent-based representations [7–11], and template-based representations [12–14] solutions have been developed to address the issue. Such declarative network operation models contrast with the traditional imperative networking model, which requires network engineers to specify the sequence of actions needed on individual network elements and creates a significant potential for error.

Intent-based approaches such as Intent Based Networking (IBN) are gaining more attention in both research and commercial communities as they effectively resolve the challenges of conventional network design with regard to flexibility, efficiency, and security [15]. IBN is generally employed to transform the business intent of a user/customer from the user's personal requirements and performance targets into network configuration, operation, and maintenance strategies, such that it captures and translates the intent into automatable network configurations that can applied consistently across the network. Typically, a business intent can be either captured through a graphical user interface (GUI) or manually by defining them using data formats such as JavaScript Object Notation (JSON) and Domain Specific Language (DSL).

In addition to system functionality, expressed via qualitative and quantitative requirements, system security must also be considered already at the design stage in order to realize a "secure by design" system that has been designed to be fundamentally secure. One approach in this area is extending the Design Space Exploration (DSE) model to cover security aspects, as done in [16, 17]. By integrating security into DSE, the authors provided a solution for verifying any potential system security issues at the design stage, a significant improvement over the traditional approach that involves manual security audits and mitigation once the system has been already designed.

This work too focuses on extending the DSE approach to take into account system security, and for this purpose we leveraged Weaver [18, 19], which is an intent-based system designer that targets IT/NW services. The main difference with other research is that Weaver adds support in DSE for specifying an abstract service requirement as input that can be used to bridge the gap between customers and system designers mentioned above, thus making it possible for Weaver to address system design in a flexible and efficient manner.

1.2 Motivation

Conventionally, systems are manually designed based on the tacit and explicit knowledge of the system designer, where the system designer often utilize checklists to validate the functional and security requirements. Although this may work for small or relatively simple systems, the tractability of a system is quickly lost for complex system of systems, especially when involving security. On top of that, changes in system requirements over time make necessary repeated re-verification to certify that the intended system still meets the required security level.

Failure to do so can have dire consequences. For example, one of the largest cybercrimes related to identity theft was attributed to American credit bureau Equifax in 2017, where it suffered a data breach due to failure to update the third-party software on their servers with the latest version. While this provided the adversary with initial access into the vulnerable servers, the main factor that caused the data breach was due to failure in the design architecture, where further access to other systems were possible due to insecure network design that lacked sufficient segmentation [20]. Even with a well-funded and well-staffed team of security experts, manual audit to changes in the design may be insufficient as addition of new components and features may change the overall security of the intended system.

Furthermore, empirical studies [21, 22] show that about half of security

issues are the consequence of architecture design weakness. To make matters worst, the fact that the systems that are set up to keep the communication infrastructure secure are designed and managed by humans. Human has always been a root of failure in IT systems, where human errors such as lack of training, intentional or unintentional mistake, lack of risk perception, and poor awareness typically leads to security breaches [23]. Automation is one of the approach of minimizing human error, especially in tasks that are repetitive and involves a large checklist. While there are some research that look into automated system design which is inline with the goal of eliminating the disadvantage of human design [7, 10, 11, 18], automation of secure design is still relatively uncommon.

The aim of this research is to improve the security of a system by introducing formal verification of the security characteristics of a system into fundamental system design, and also through practical experiment verification. This research proposes an automated secure design framework, SecureWeaver, and its implementation for networked systems to achieve this goal. For an automated system designer to be able to validate whether a design is secure, a knowledge base that embodies the tacit and explicit knowledge of secure design is necessary. Using it, formal verification algorithms can be introduced to verify the generated system design according to the threat and mitigation information in the knowledge base. Even with the ability to automatically create a secure system design, it cannot be said that the resulting system is secure. Inconsistency throughout the life-cycle of the system may cause it to deviate from the designed secure architecture, hence justifying the need for a secure implementation at the same time.

1.3 Main Contributions

This thesis presents the individual components and processes needed to realize an automated secure system design framework and its implementation for networked systems, SecureWeaver. The following are the main contributions:

1. A secure design knowledge base for SecureWeaver that encompasses a database of secure refinement rules for the proposed automated secure system designer, a MITRE ATT&CK-based threat mitigation knowledge base, and a database for secure protocols.
2. The SecureWeaver framework that supports the creation of a secure system design for both information technology/network (IT/NW) and IoT systems; the framework accepts an abstract intent with qualitative, quantitative and security requirements as input, and refines it into a secure system design.

3. Two case studies on secure end-to-end communication and secure configuration that demonstrate the challenges of implementing secure networked systems; these studies guided the process of modelling threats and system components used for the evaluation of SecureWeaver.
4. A set of models for IT/NW and IoT system design via SecureWeaver, such as a corporate network and IoT appliance along with hardware-level design that make it possible to automatically design this kind of systems in a secure manner.

The main impact of this research is that it will make possible to decrease the human factor drawback in system design by automatically designing secure system with the proposed framework. By using formal methods for security, this will add to the knowledge field in automated system design. Furthermore, this research involves a joint-research collaboration with NEC in introducing security into their existing system designer; successive patent applications were made from the results of this research, and there is potential use commercially by the company, thus leading to a high impact in society.

1.4 Thesis Outline

The thesis is organized as follows.

- Chapter 1 introduces the overview, motivation, main contributions, and outline of the thesis.
- Chapter 2 introduces the related works of the thesis, such as cybersecurity knowledge databases, intent-based design, secure system design, IoT design and provisioning. This chapter also introduce the state-of-art intent-based system configuration designer, Weaver, which was used as basis to realize the implementation of automated secure system design presented in the dissertation. Moreover, existing IoT development and management platforms are also introduced in this chapter.
- Chapter 3 presents the philosophy behind automated secure system design, where trustworthiness and its characteristics are discussed. The methodology of automated secure system design is also introduced, and its overview, secure system designer requirements and system implementation are discussed. Furthermore, preliminary works on automated secure design are also presented in this chapter, such as system specification verification with Satisfiability Modulo Theories (SMT), system design ontology, and the evaluation of existing IoT platforms.

- Chapter 4 presents the secure design database, covering the secure design threats and rules, threat mitigation knowledge base, and the ontology extension of the proposed knowledge base. The definitions of security threats, logical and conceptual connections, and security refinement rules are covered in this chapter along with the MITRE ATT&CK-based threat mitigation knowledge base.
- Chapter 5 details the automated secure system designer, SecureWeaver. The overview of its mechanism and a detailed explanation of the security verification functions based on the network domain in MITRE ATT&CK Enterprise matrix are presented.
- Chapter 6 presents the reference IoT hardware platform, MkIoT. The requirements to be considered when designing an IoT device are introduced, along with details on MkIoT based on the stated requirements. Besides that, three case studies are presented in this chapter, where one implements a secure end-to-end IoT application, another one implements hardware-based security on the IoT device, and lastly the IoT life-cycle management is discussed.
- Chapter 7 presents the evaluation of the proposed automated secure system designer in this thesis. Three case studies are presented in this chapter, where the first one evaluates SecureWeaver from a secure corporate network design perspective, and is by a case study on secure IoT appliance system design. The final case study evaluates SecureWeaver for secure IoT hardware system design. A feature evaluation and comparison are also discussed in this chapter.
- Chapter 8 presents the conclusion of the thesis and outlines future research directions.

Chapter 2

Research Background

In this chapter, a comprehensive review of the fundamental knowledge and state-of-the-art works related to this research are explored. First, related works in cybersecurity threat mitigation knowledge base are discussed, followed by related works in networked systems, both Information Technology and Network (IT/NW) and Internet-of-Things (IoT), respectively. Related works in intent-based IT/NW design automation and secure intent-based design are reviewed, followed by a detailed review of a DSE based IT/NW system designer, Weaver, respectively. Furthermore, related works in automation of IoT design and provisioning are reviewed along with specific works in IoT security. Finally, this chapter closes with an evaluation of existing IoT platforms.

2.1 Cybersecurity Knowledge Databases

The work in [24] presents an approach to create a cyber threat dictionary by mapping the ATT&CK Industrial Control Systems (ICS) matrix into the Facility Cybersecurity Framework (FCF) cybersecurity assessment tool. Both proactive and reactive measures can be done by utilizing the mapping provided by the cyber threat dictionary threats and their mitigations. While this work shares the similar concept of using a cybersecurity attack and defense framework for mitigating a threat, SecureWeaver further introduces a verification mechanism to enable the automatic verification of the applied mitigations.

The work in [25] presents a methodology to unify both MITRE ATT&CK Enterprise and Industrial ICS (Industrial Control Systems) to provide a broader and holistic mitigation coverage together with the Cyber-Security Culture model. This enables one to identify the security vulnerability using a structured evaluation methodology from an organisation security assessment result. This work also presents simple and complex use case scenarios to illustrate the mapping of MITRE ATT&CK techniques and mitigations, and Cyber-Security Culture model classification of the assets to each threat and

the relevant mitigations.

The work in [26] presents a semantic model of cyberattacks and vulnerabilities based on Common Attack Pattern Enumeration and Classification (CAPEC) and Common Weakness Enumeration (CWE) knowledge base. The ontology model links CWE to other attributes and CAPEC via relevant relationship, which is implemented using OWL (Web Ontology Language) that enables query capabilities to obtain classification of the security concepts.

The work in [27] presents a cybersecurity threat modeling framework based on MITRE ATT&CK Enterprise matrix. The Meta Attack Language (MAL) framework is used in tandem with MITRE ATT&CK tactics and techniques to describe the system assets, possible attack sequences and their mitigations. With this approach, simulation of the IT system is done with varying security settings and system architectural design to obtain a secure system implementation.

The work in [28] presents a unique approach to enable bidirectional mapping between multiple third-party security databases, such as MITRE ATT&CK, NIST Common Weakness Enumerations (CWE), Common Vulnerabilities and Exposures (CVE), as well as Common Attack Pattern Enumeration and Classification (CAPEC). This illustrates the feasibility of expanding the threat mitigation knowledge base in this thesis beyond the scope of MITRE ATT&CK to cover other third party databases.

2.2 Intent-Based Design

The work in [9] presents a framework for Software-Defined Network (SDN) applications that is an intent-based Northbound Interface (NBI). The framework is able to translates the objectives, requirement and constraints of an application without requiring the usage of network-specific language in the intent or comprehension of the underlying network in an SDN environment which supports the application. While this framework specifically targeted the SDN domain, the core concept behind its intent-based "translation" is similar to the automated system design approach in this thesis.

The work in [29] also presents a framework for OpenFlow/IoT SDN domains, which is also an intent-based NBI. The framework is capable of provisioning the network path given the constraints such as end-to-end Quality-of-Service (QoS) by using a service chaining description as intent. This framework targets both the IT/NW and IoT domains. The main difference with the work in this thesis is that the framework is intended for the management of network paths in existing SDN environments, whereas

this thesis approach is able to design a networked system from scratch.

The work in [11] utilizes natural language to describe the intent in a network management framework. The framework first accepts an intent that is described using natural language from the user and extracts the requisite information using machine learning algorithm. The extracted information is then fed into an intent assembly module. This module generates concrete network configuration commands that satisfies the intent requirements. This work does not meet the goal of this thesis of automatic secure system design as it is an intent-based network management tool for existing network infrastructure.

Existing automated system design frameworks such as [7, 8] are capable of designing a system are normally targeted toward either architecture or parameter-level design platforms for their specific domains. The requirements and entities of a topology graph (e.g., components, relationships and constraints) are typically specified in architecture-level design, whereas in parameter-level design, the parameters and fine tuning are specified according to the given topology. For example, parameter-level design tools like [7] have a high level of flexibility compared to their architectural-level design equivalents. Still, they are incompatible with the intended use cases in this thesis, which is architecture-level design for networked systems covering both the IT/NW and IoT domains.

2.3 Secure Intent-Based Design

The work in [30] presents an intent-based framework for Network Function Virtualization (NFV), where the framework models and computes the suitable virtual network functions (VNF) chains to meet the intent requirements. This framework requires the abstract weights for the non-functional requirements to be predefined before performing quantitative computation. The resulting quantitative scores of the VNFs are then clustered to their corresponding levels to obtain the most compatible VNF that meet the given requirements. This approach is not as flexible as the approach for architecture design in this thesis.

The work in [31] presents a model-driven design for cloud services orchestration that also incorporate security-level evaluation. A template-based matching method is utilized to meet the required intent. This work also employed numerical calculation approach to evaluate security propagation in a topology graph. This is done to verify whether the output topology meets the security level specified in the intent. Alternatively, the approach in this thesis specifically links the security threats to their corresponding mitigations

by consulting a security knowledge base that is based on MITRE ATT&CK Enterprise data. This is much more concrete than an abstract quantitative assessment.

The work in [10] presents a framework, named IBCS (Intent-Based Cloud Services for Security Applications), which is an intent-based security service automation framework for cloud environments. The framework parse the user provided network security intent and translates it into concrete configurable security policies. This is achieved by first extracting the context data via the Deterministic Finite Automata (DFA) method before linking the context to a suitable Network Security Function (NSF). The extracted data is then converted into security policies using the Context-Free Grammar (CFG) technique, where the resulting output configuration can be used to configure relevant network interfaces. The main difference of this work with the thesis framework is that IBCS only computes security policies that are to be applied in an existing cloud environment, whereas the thesis framework designs a secure system architecture from scratch to satisfy the given intent and requirements.

The work in [32] presents a DSE-based system designer that targets application domains such as IoT and cyber-physical systems (CPS). The work utilizes DSE techniques to refine a secure embedded system topology. While this work is specifically used to design the hardware and software properties of IoT device, this thesis approach mainly address networked system architecture design issues. Moreover, this work utilize Microsoft's STRIDE threat model to derive its attack types, whereas this thesis approach utilizes MITRE ATT&CK Enterprise data for threat and mitigation representation in the system design.

2.4 Weaver: Intent-based System Configuration Designer

The secure system designer implementation in this thesis is based on the Weaver IT/NW system designer [18,19]. An overview of Weaver's processing flow is presented in Fig. 2.1. Each rectangle in the figure represents the state of the topology at a particular step, and the red arrows in the pyramid are the rule-matching refinements performed by Weaver.

First, a service configuration which contains abstract and/or concrete entities is input into Weaver. An entity is said to be concrete if its type for both components and relationships is definite (e.g., Red Hat Enterprise Linux OS, or HTTPS connection). Otherwise, it is said to be abstract (such

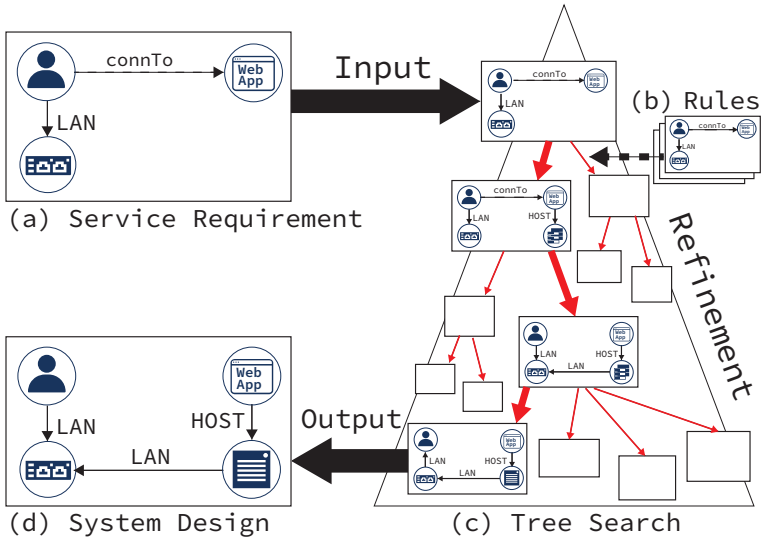

Figure 2.1: Original Weaver system designer processing flow.

as a generic OS or network connection). These abstract entities are then continually refined using tree search with matching refinement rules until the final topology state does not contain any abstract entities. The final topology, which is fully concretized, is then output by Weaver as the system design corresponding to the given service requirement. Weaver's data format, rules and topology refinement and the tree search approach are briefly described in the following subsections.

2.4.1 Data Format Definitions

Weaver's data format is structured similarly to TOSCA (Topology and Orchestration Specification for Cloud Applications) [33], a specification that declaratively describes a service configuration in a topology to enable provisioning automation via the definition of components, relationships, and their attributes.

A component is defined as a pair "$v : ctype$", where v is the component identifier and $ctype$ is its type. A component type is defined as $ctype = (name, abs, cap, req)$, where $name$ is the $ctype$ name, abs is its abstractness, cap is the component capability, and req is its associated requirement, describing one or more relationships with other components.

An edge, e, is also known as a relationship between two components in Weaver, where $e = (v_{src}, v_{dst}, rtype)$ is a triplet of the source component identifier, destination component identifier, and its relationship type. A rela-

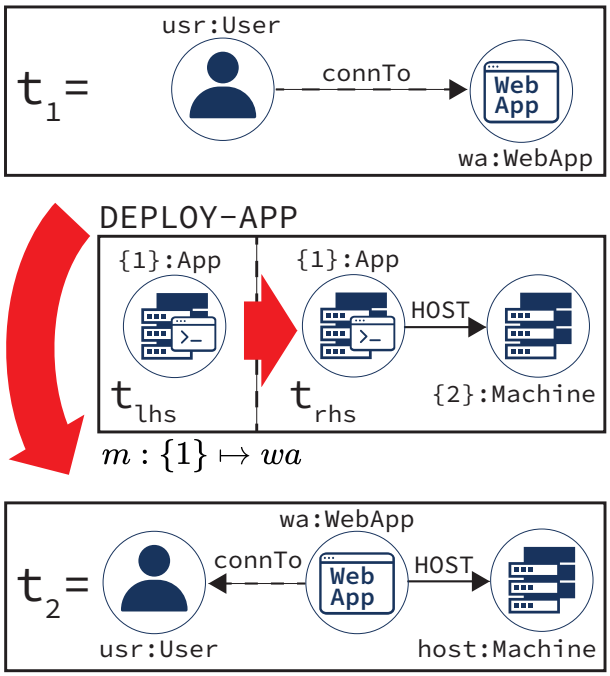

Figure 2.2: Example of the application of the rule DEPLOY-APP as a Weaver topology refinement step.

tionship type is defined as $rtype = (name, abs)$, where $name$ is an $rtype$ name and abs is its abstractness. For example, a non-specific operating system (OS) is an abstract component, while "Windows" is a concrete component that can be derived from an abstract OS component via inheritance.

A topology can be formalised as the tuple $t = (V, E)$, where $V = \{v_{nid1}, ..., v_{nidn}\}$ and $E = \{e_{eid1}, ..., e_{eidn}\}$ are a set of components and relationships.

2.4.2 Rules and Topology Refinement

To refine an abstract service requirement into a completely concrete system topology, the refinement process is performed iteratively to transform a topology from one state to another using matched refinement rule. A refinement rule, r, is a one-step refinement process that is denoted as a tuple $r = (t_{lhs}, t_{rhs})$, where t_{lhs} is the left-hand side and t_{rhs} is the right-hand side of the rule. An illustration of a rule, DEPLOY-APP, is shown in Fig. 2.2. Generally, a match is a mapping from the component placeholders, $\{1\}, ..., \{n\}$ to component identifiers $nid_{\{1\}}, ..., nid_{\{n\}}$, where $m(\{i\}) = nid_{\{i\}}$.

An action, $r[m]$, is a function where r is provided with a matching, m to load the rule component placeholders with the relevant component identifiers found in the topology. As an example, the web application wa in topology t_1 in Fig. 2.2 is transformed into the topology t_2 by the rule `DEPLOY-APP` with the match pattern, $m : \{1\} \mapsto wa$. An arrow with dashed line represents an abstract relationship, while an arrow with solid line represents a concrete relationship. The base component *App* in Fig. 2.2 has an inheritance relationship with the $WebApp$ component type. Hence, component $\{1\}$ of type *App* is equivalent to wa of type $WebApp$, such that $\{1\}$ in the rule `DEPLOY-APP` can be replaced by $\{1\} \overset{HOST}{\longrightarrow} \{2\}$.

2.4.3 Tree Search-based Algorithm for DSE

Weaver finds refinement candidates through a deterministic search process, by which it iteratively applies relevant actions to obtain a completely concrete topology. The search algorithm exits when all the following conditions are met for a given topology $t = (V, E)$:

- All components $v \in V$ are concrete
- All relationships $e \in E$ are concrete
- All requirements of a component are matched by the capabilities of another component that is connected to it via a relationship

If the exit condition is met, the selected system topology is said to be concrete. More detailed information on the tree search algorithm and the definition of topology concreteness can be referred in [19].

2.5 IoT Design and Provisioning

This section presents related works in IoT system design, as well as its secure system design counterpart. Related works on frameworks for automated provisioning of IoT systems are also presented.

2.5.1 IoT Design

The work in [34] outlines the directions for improving development of IoT, which mainly highlights the importance of scalable models of interactions between IoT entities. The use case scenarios illustrate the mapping of the requirements to intents, where each entity and their relationships have structure similar to an ontology. The work also discusses various issues on

simulation implementation such as entities integration, cloud platform and tools, with the final goal of validating real life IoT scenarios in a network.

The work in [35] discusses the understanding of IoT system from broader IoT architecture reference ontology such as IoT Architectural Reference Model (IoT-ARM). This includes the structural definitions of devices, semantic understanding of flows, services and quality measures within and between entities in the system from the IoT-ARM perspective.

The work in [36] presents an approach for the verification and validation of the software of the "Things" composing IoT systems using the thing-in-the-loop approach. The work utilizes context models to safely test various scenarios using simulated environment. Besides, the topology is expressed using Unified Modeling Language (UML) in tandem with state machine diagrams.

The work in [37] presents a framework for IoT Testing as a Service (IoT-TaaS). The framework in this work provides remote interoperability testing as well as scalable conformance and semantic validation in its testing. The framework is core focus is on protocol conformance testing, as it supports various protocols in a modular manner with automated conformance testing. The semantic testing in the framework is limited to testing whether the protocol data/message are syntactically and semantically correct.

The work in [38] presents a general classification of IoT devices based on the NIST's Network of "Things" framework [39]. This work also proposes a taxonomy that is employed for classification of IoT devices, as well as example scenarios that demonstrates its ability to classify common tools, protocols and APIs that can be employed to standardize IoT implementations. The work also demonstrates a use case where it uses the taxonomy to profile the correct operation of a model for quality assurance testing and intrusion detection.

The work in [40] presents a design methodology for end-to-end security implementation that involves IoT devices and its validation using the digital twin concept. Both black box and white box modeling are employed in this work for modeling the global functionality of the end IoT device and its end-to-end network path in the simulation.

2.5.2 IoT Secure Design

The work in [41] presents a framework that allows a designer to model the functionality of the system, available architecture component in the system and security attack scenarios. By analysing the task graph, the security attack scenarios can be modeled with respect to the system architecture via attack graph and risk tree of an IoT system architecture.

The work in [42] presents a detailed analysis of Arduino device security, as well as the demonstration of vulnerability exploits for those devices. The work describes various exploits against common microcontrollers, where the work also demonstrated heap buffer and stack buffer overflow.

The work in [43] presents a methodology on performing threat analysis on IoT device, especially Arduino. The threat analysis in the study is conducted according to the Common Configuration Scoring System (CCSS) assessment from NIST. The base score from CCSS is employed especially in optional temporal and environmental perspectives to assess the security posture of a typical IoT system. The work also includes a case study on Arduino vulnerability impact on IoT devices.

2.5.3 IoT Provisioning

The work in [44] presents a framework that utilized the Service Oriented Device Architecture (SODA) approach where the device is encapsulated as a device service to offer data on a higher semantic level to services. The work employed both JSON and REST API for the communication between the components in a system, which is similar to Topology and Orchestration Specification for Cloud Applications (TOSCA).

The work in [45] presents an automated orchestration framework of IoT services providing dynamic resource provisioning and automated application deployment in fog computing architecture. The intent requirement of the framework specifies parameters such as priority, computation, latency, privacy, and output. This work also includes various system components in the framework such as version control server, continuous integration tool, and container registry, as well as IoT device and an orchestration server that is powered by the cloud service provider IBM Watson IoT.

2.6 IoT Development and Management Platforms

In this section, several IoT platforms are introduced as a part of the system specification study during the preliminary work stage of this thesis.

First, an IoT platform in this thesis is defined as a suite of software components providing capabilities like visualization, data archiving and analytics of information from IoT devices and equipment [46]. There are many commercial IoT platforms such as ThingSpeak, SensorCloud, thethings.io, and many more. However, many of these platforms are mainly for data visualization, lacking crucial feature such as IoT life-cycle management. In this

section, the existing IoT platforms that features IoT life-cycle management are evaluated. The IoT platforms that are evaluated are:

- Bosch IoT Suite [47]
- IBM Watson IoT Platform [48]
- Microtronics IoT Suite [49]
- IoT Suite [50]

The detailed analysis of these platforms is presented in Section 3.3.3.2. IoT platforms such as Bosch IoT Suite, IBM Watson IoT Platform and Microtronics IoT Suite are commercial products, while IoT Suite is an academic research.

Bosch IoT Suite is an IoT platform that provides three main pre-integrated service packages: (i) asset communication; (ii) device management; and (iii) software updates. These pre-integrated service packages are built up from individual services such as IoT hub, remote manager, IoT Insights, IoT Rollouts, and IoT Things. For example, the asset communication pre-integrated service is an integration of IoT hub and IoT Things services. Bosch IoT Suite provides user with the ability to enroll their IoT device into the management platform, and use its pre-integrated services or individual services to cater to their requirements.

Besides that, IBM offers the IBM Watson IoT Platform as one of the many services in IBM cloud. Users can typically use the IBM Watson IoT Platform as a way to enroll IoT device into IBM cloud, and utilize the vast services in IBM cloud such as business analytic, machine learning to DevOps services. This provides the user with a wide range of option to build their IoT solution on the platform.

Microtronics IoT Suite, on the other hand is different than the first two platforms introduced in this section. Microtronics IoT Suite main function is to allow users to create IoT application on-the-fly with the company's proprietary hardware solution. This allows users to quickly build and test their IoT application as a proof of concept. The platform also provides various sample templates such as datalogger with end-to-end cloud connectivity, with dashboard included. User can implement their application on Microtronics's web integrated development environment (IDE), which automatically build, setup and deployment the application.

The last IoT platform in the list is the IoT Suite, which is part an academic project. The IoT Suite is a toolkit for prototyping IoT applications, where it aims to reduce development efforts at various stages of a typical IoT application development pipeline. The IoT Suite provides a compiler that compiles a relatively abstract definition of an IoT application into a concrete application, where the result is passed to a deployment module

for deployment. The IoT Suite also features a runtime system for the deployed code to run on. The source code for the IoT Suite can be found on (https://github.com/pankeshpatel/IoTSuite).

2.7 Summary

This chapter presented a comprehensive review of various works and fundamental knowledge related to this thesis. Related works in cybersecurity threat mitigation knowledge base framework and works in networked system were discussed. This was followed by a discussion on related works in intent-based IT/NW design automation and secure intent-based design. Furthermore, a detailed review of Weaver, a DSE based IT/NW system designer that is used as the underlying system designer in this thesis was presented. Related works in automation of IoT design, its provisioning and specific works in IoT security such as Arduino were also discussed. Finally, the chapter ended with an introduction of existing IoT platforms that will be later evaluated in Chapter 3.

Chapter 3

Philosophy of Secure System Design

In this chapter, the philosophy of secure system design of this thesis is presented. First, the topic of trustworthiness in systems is introduced, followed by its characteristics such as safety, security, safety, privacy, reliability, and resilience.

Next, the methodology of automated secure system design is introduced, where its requirements are presented. This chapter also presents the preliminary work on automated secure design on IoT, where the formal verification of the system specification using Satisfiability Modulo Theories (SMT), ontology based on IoT system design, and the system specification study of existing IoT platforms are described in detail.

3.1 Why Do We Need Secure Design?

System designs and their corresponding security architectures typically involve a good deal of concrete and technical aspects, where most aspects of ***how*** to accomplish the objectives are likely to be of technical nature [51]. On the other hand, subject matters concerning the ***why*** does not need to be technical, especially for the intent of a security architecture on a systems-level perspective. The conceptual intents tend be characterized by various non-technical aspects such as compliance requirements, moral, ethics, and social code [52, 53].

The difference between verification and validation is analogous to the difference between technical and conceptual aspects:

- Verification: are we building the system right?
- Validation: are we building the right system?

Applying the same concept to the technical and conceptual aspects, it is counterproductive to do something correctly while one is building the wrong thing in the first place. Hence, a philosophical stance is required with

regards to secure design. To delve further into this, we look further into trustworthiness and other aspects such as security, safety and privacy. The topic of trustworthiness is a part of the publication [54].

3.1.1 Trustworthiness

Modern ICT has become more integrated in everyday life, and the future society will be increasingly dependent on ICT systems. The issue of how much the ICT systems can be trusted is critical, as failure or malfunction of systems especially in safe and secure operations of critical infrastructures can become a life threatening situation. The issue with trustworthiness is that the term is overloaded, especially when heavily used in most marketing materials. First, the following contexts on trust are listed:

- Trust is relative
- Definition of trustworthiness may be different or evolve in time

Trust is firstly relative as it is a human trait, for example, today I trust you more, tomorrow I trust you less. This shows that trust is not a static property or characteristic that can be easily defined. Secondly, the definition of trustworthiness may changes over time. This is illustrated in Fig. 3.1, where different stages of the industry revolution have different characteristics that embody trustworthiness. In the early industrial age, reliability was the key characteristic of trustworthiness, as most designs were not reliable, where as over time, the reliability of systems has improved and stakeholders in the industry began to address other characteristics [1].

The definition of trustworthiness will be discussed below, especially for the IoT domain. To the best of the author's knowledge, the issue of trustworthiness, especially in the context of IoT domain, has only been addressed by few organizations (e.g., International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) committee "Internet of things and digital twin" JTC 1/SC 41 [55], the National Institute of Standards and Technology (NIST) Cybersecurity for IoT Program [56], and the Industrial Internet Consortium (IIC) [57].

The work in [58] by NIST discussed trustworthiness of Cyber-Physical Systems (CPS), where security, privacy, safety, reliability, and resilience are indicated to be characteristics of trustworthiness. However, the trustworthiness concept was not explicitly defined in that work. On the other hand, the IIC in [59] has performed a more extensive analysis of IoT trustworthiness, and the definition it proposed is the one we consider as the best available at present:

"Trustworthiness is the degree of confidence one has that the system

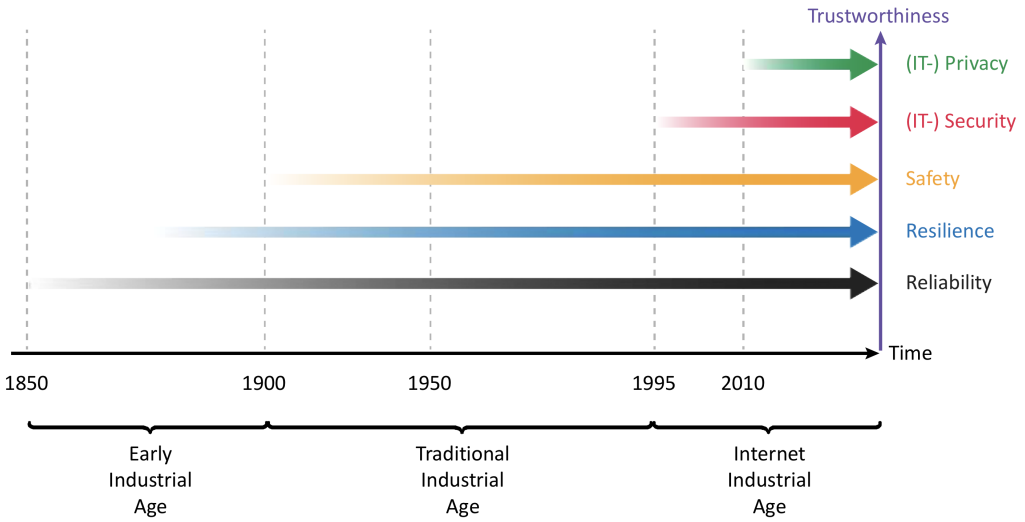

Figure 3.1: Evolution of trustworthiness in industrial systems [1].

performs as expected with characteristics including safety, security, privacy, reliability and resilience in the face of environmental disturbances, human errors, system faults, and attacks."

While it is difficult to encompass all aspects of trustworthiness in this definition, this thesis presents the works that focus on the aspect of security. The following subsection will discuss the components of trustworthiness.

3.1.2 Safety, Security, Privacy, Reliability, and Resilience

The necessity for safety, security, privacy, reliability, and resilience is important in modern ICT, especially in the IoT domain. While the safety, privacy, reliability, and resilience are not part of the scope in the thesis, strong security is a prerequisite for both safety and privacy, thus putting emphasis on the importance of an effective and comprehensive security architecture in a system design. Each of the trustworthiness characteristics are described next.

Safety Issues regarding safety are generally tackled via laws, regulations and standards. This typically requires verifying that the particular system satisfies the requirements of the relevant laws, regulations and standards, both during the design phase and system operation. These requirements are

commonly specific to the target domain/industry of a particular country, such as in the automotive industry, where an international standard on road vehicle functional safety was published by ISO (International Organization for Standardization) [60], and for domestic standard, a framework on self-driving safety assessment was published by Japan Automobile Manufacturers Association (JAMA) for autonomous functionality in Japan [61].

Security The information security "CIA" triad is commonly used to analyze IT security issues, where its three components are Confidentiality, Integrity, and Availability. Moreover, guidelines on secure IT/IoT development have been published by various organizations and governments as suggestions on implementing necessary security mechanisms in the system, such as development guidelines from European Union Agency for Cybersecurity (ENISA) [62] and NIST cybersecurity requirement catalogs for IoT device [63]. The verification of the security component of trustworthiness comprises an extensive range of approaches. This may vary from solely ensuring that the relevant guidelines have been adhered to, all the way to static/dynamic source code analysis.

Privacy The main focus of privacy analysis is generally on the collected data by an IT/IoT (e.g., via API/sensors), and on how that data is transmitted and stored (e.g., on premise/in the cloud). The objective of privacy analysis is to verify whether the system and its end-to-end components are compliance with appropriate regulations. These may involves regulations like the Health Insurance Portability and Accountability Act (HIPAA) [64] for health-related applications in the U.S. and the General Data Protection Regulation (GDPR) in the EU [65]. The verification of the privacy component of trustworthiness can be practically performed via evaluation during system design phases and operation.

Reliability Reliability theory is an established field, with its widely used reliability metrics such as the mean time between failures (MTBF) or mean time to failure (MTTF) especially for hardware that may malfunction as due to component or material failure [66]. On the other hand for systems related to IT/IoT, software reliability has to be taken into account in tandem with hardware reliability.

Resilience The concept of resilience as a component of trustworthiness is highly dependent on the context, such that it may depends on parameters such as the system architecture, its operational environment, and the type

of the disruptive event [67]. One approach that is often used to enhance the resilience of a system is via redundancy. However, redundancy must be coupled with the elimination of single points of failure and good maintenance for it to work effectively [68]. Besides that, characteristics such as graceful degradation is also crucial from a resilience perspective in order to make sure that misbehaviour of a system does not adversely affect the other components of trustworthiness.

3.2 Methodology of Automated Secure System Design

This section introduces the methodology for automated secure system design and its implementation for the IoT domain.

3.2.1 Methodology Overview

During the architecture design phase of a given system, both functional and non-functional requirements, which include the security of the intended system, must be captured. Security in this context refers to the protection of a system, and the requirements of a secure system architecture are mainly related to how much uncertainty and risk one is willing to tolerate [51]. While no system can be made perfectly secure, having an inadequately secured system has negative consequences. Thus, an automated secure system designer should be able to have a quantifiable security level target, and be able to quantitatively discern whether a particular design is secure enough in terms of that target.

An automated secure system designer should be able to apply proactive measures, which are measures that are directly applicable in a system design, such as deploying a firewall where needed to filter the incoming traffic. Reactive measures include actions such as cyber threat incident detection, response, and recovery efforts. Such measures cannot be concretely specified at design phase, hence they are to be applied in the form of assumptions that must then be put into application during the actual system deployment.

There are many methods to secure a system, such as following best-practice guidelines or threat modeling. When handling a security threat, there may be more than one approach to address it, such as removing the threat (prevention), reducing the impact of the threat (mitigation), or even transferring the risk via insurance or passing on the risk to the end-user [51].

Furthermore, generating a secure design does not guarantee that the design will be implemented correctly. Erroneous implementation may lead

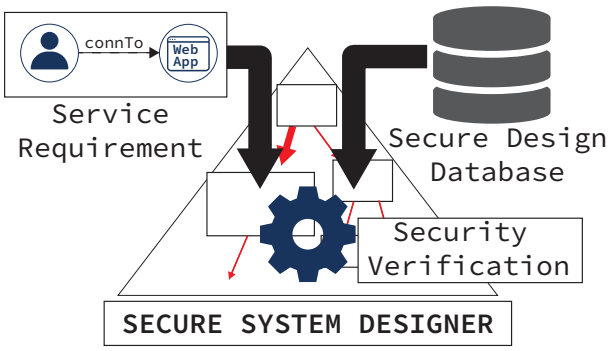

Figure 3.2: Overview of the secure system designer SecureWeaver.

to deviations from the original intended design. Ideally, the automated secure system designer would output a system design, and the output will be automatically build and deployed. This would be similar to the software world's DevOps CI/CD (continuous integration/continuous deployment) process. However, we consider the automated implementation to be out of the scope of this thesis.

In this work we propose an automated secure system designer that is able to accept a service requirement as an input, including security aspects, then process the input and generate a system design that is both concrete and secure. In order to achieve this goal, the system needs to consult a database that contains both design patterns and security knowledge. Hence, the key requirements for such a system are the following:

1. Ability to design a system in an automatic manner based on input qualitative and quantitative requirements (see Section 2.4)
2. Use of a secure design database with relevant design patterns and knowledge that make possible the secure system design (see Chapter 4)
3. Use of a security verification mechanism to automatically check that the generated system design satisfies appropriately the input security requirements (see Chapter 5)

This methodology is illustrated in Fig. 3.2, which gives an overview of our implementation, SecureWeaver. Note how the service requirement and secure design database are used by the secure system designer that employs the security verification mechanism to verify the system design candidates.

A service requirement is the top-level description of the intended design outcome (intent), which is inspired by the concept of intent-based networking (IBN). The service requirement can be either completely conceptual (equivalent to designing a system from scratch), or it can be an incremental addition to an existing network configuration. The creation of the service

requirement is typically a two stage process: (i) input requirement capture, and (ii) composing the service requirement according to SecureWeaver's framework. The service requirements must first be captured from users of the networked system that is to be designed, such as external customers planning to deploy a network from scratch, or an internal department in an organization planning to deploy a new service for their department on an existing network infrastructure. Hence, technical staff will then express these requirements to create the actual content inside the intent file in a format that is compatible with SecureWeaver. The technical staff may be:

- Designers such as system/security/solution architect, etc.
- Actual operators such as system/cloud administrator, etc.

Components that describe abstract or concrete parts of the intent and the relationships that denote communication or component relations are part of a typical service requirement. However, a system designer is not able to derive an appropriate system design when the specified threats and relationships in a service requirement lack information and context.

Hence, a certain level of context is required in order to accurately express the security requirements and derive a suitable solution that: (i) meets all the functional requirements; (ii) has no unmitigated security issues. In this manner, we can explicitly describe a security threat on any of the component and relationship in the service requirement, where the automated secure system designer must take into account during its refinement process.

For the automated secure system designer to be able to interpret the defined threat, it must refer to a database to obtain the required information and act accordingly. This is where the secure design database comes into play. The secure design database should contain information, such as secure design patterns, security threats and their corresponding mitigations, and other related data needed for informed decision making.

To realise the vision of such an automated secure system designer, we built our prototype implementation upon an existing automated system designer, named Weaver. By integrating a security knowledge base into Weaver with relevant modifications, we made it possible for the system designer to consult the knowledge base regarding entities in its topology for security issues and mitigate them with appropriate mitigation techniques. Moreover, we also had to extend Weaver to accommodate security capabilities, especially in the areas regarding the components and their relationships.

3.3 Preliminary Work on Automated Secure Design

This section presents the preliminary work we conducted on automated secure design in the IoT domain. The first subsection explores the usage of SMT to perform formal verification of IoT system specifications, such as boolean logic and quantitative constraints. This is followed by the preliminary work to expand the basic IoT "vocabulary" in the first subsection to an ontology framework that allows extensible semantics for heterogeneous IoT domain.

3.3.1 System Specification Verification with Satisfiability Modulo Theories (SMT)

To formally verify a system specification, a verification framework is required to be able to take arbitrary format (e.g., integers, boolean, logical connective, etc.) and compute an output whether the following input is valid. One of the most commonly used verification frameworks are the Satisfiability Modulo Theories (SMT) and Satisfiability (SAT) solver. Both SMT and SAT solvers are typically used to solve huge systems of equations, or in another terms, constraints in terms of system requirements in the specification. The main difference between SMT and SAT solver is that the SMT solvers are the "front-end" of the SAT solver. SAT solvers are limited to boolean equation in conjunctive normal form (CNF), while SMT is able to take a system of equations as an input in arbitrary format and process them using underlying SAT solvers.

The high-performance theorem prover Z3, developed at Microsoft Research is utilized in the preliminary work as a proof-of-concept. The Z3 solver is also an open source and cross-platform SMT solver, where it provides bindings for languages such as C, C++, .NET, Java, OCaml, Python, Julia, and WebAssembly. Before implementing a prototype verification framework for IoT system specification, the work in [2] is adapted to built the initial implementation of boolean-only verification. The Optimal Package Install/Uninstall Manager (OPIUM) by [2] is an improved Linux package manager that solves the dependencies and conflicts constraints using a SMT solver. The package manager dependencies and conflicts problem is illustrated in Fig. 3.3, where the individual packages (`a`, `b`, `c`, `d`, `e`, `f`, `g`, `y` and `z`) dependencies and conflicts are shown (`d` and `e`).

The boolean verification of package dependencies and conflicts can be adapted to describe the dependencies and conflicts in the IoT system re-

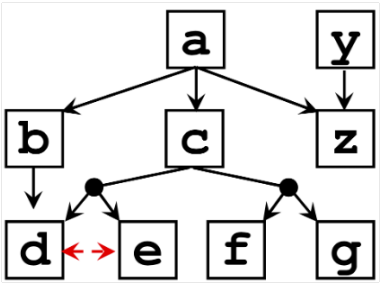

Figure 3.3: Example of Linux package manager dependencies and conflicts [2].

quirements. However, IoT system specification verification under boolean-only scenarios are too limited, as many of its requirements are quantifiable constraints, such as IoT device's power budget for battery operation. Hence, both boolean and quantitative constraints are considered in the proof-of-concept in this preliminary work for automated secure design.

First, an IoT design scenario is assumed, where a battery power budget and functionality verification is evaluated. The unit watt (milliwatt for low-power IoT devices) is the right term used to describe power consumption by a device. However, the current consumption measured in milliampere is all too often used instead. This mainly corresponds to battery-powered applications, where a tiny battery may only provide a limited max current output without damaging the battery. Hence, the current consumption will be used in our work for the power budget verification. A basic JSON dictionary consisting of various IoT hardware features, such as the wireless capabilities with their typical power consumption are shown in Code 3.1 under the variable "dict". The dictionary features various microcontroller unit (MCU) such as Espressif ESP8266 and ESP32, and ATMEL ATMega32U4, each with their features as listed below:

- `power`: Typical max current consumption in milliampere
- `security`: Availability of security feature as a boolean value
- `freq`: CPU frequency speed in MHz
- `network`: List of wireless network interfaces
- `interface`: List of wired interfaces

The dictionary also features the wireless interfaces with their parameters such as power, security, and the maximum size of data in a packet. Definition of functionalities and their dependencies are also listed in the dictionary.

Since the energy consumption requirement is one of the main factors in determining an IoT device architecture for both hardware and software [69],

the quantitative verification scenario will focus on a power budget verification. Two functions are implemented to describe both dependency and conflict relationship, as shown in Code 3.1 `dependsOn()` and `conflict()`. The `dependsOn()` function implements the logical implication of an input to its dependencies, as shown from Line 19 to 23. On the other hand, the `conflict()` function implements a `NOT` logical operator between all the elements in the input, as shown in Line 26.

Code 3.1: IoT hardware verification example using Z3 in Python 3.

```
1  from z3 import *
2
3  dict = {'wireless': {
4              'wifi': {'power': 240, 'security': True, 'psize':
                   2312},
5              'ble': {'power': 13, 'security': True, 'psize':
                   242},
6              'sigfox': {'power': 100, 'security': True, 'psize':
                   12},
7              'zigbee': {'power': 40, 'security': True, 'psize':
                   128},
8              'nrf24': {'power': 16, 'security': False, 'psize':
                   32}},
9          'mcu': {
10             'esp8266': {'power': 400, 'security': True, 'freq':
                   160, 'network': ['wifi', 'ble'], 'interface': ['
                   i2c', 'spi', 'uart', 'i2s', '1wire']},
11             'esp32': {'power': 240, 'security': True, 'freq':
                   240, 'network': ['wifi', 'ble'], 'interface': ['
                   i2c', 'spi', 'uart', 'i2s', '1wire']},
12             'atmega32u4': {'power': 27, 'security': False, 'freq
                   ': 16, 'network': [], 'interface': ['i2c', 'spi',
                    'uart', '1wire', 'usb']}},
13         'functionality' : {
14              'audio': {'interface': 'i2s'},
15              'video': {'psize': 1000, 'security': True, 'freq':
                    200}
16     }
17 }
18
19 def dependsOn(char, deps):
20     if is_expr(deps):
21         return Implies(char, deps)
22     else:
23         return And([ Implies(char, dep) for dep in deps ])
24
25 def conflict(*chars):
26     return Or([ Not(char) for char in chars ])
27
```

```
28 def check(*problem):
29     s = Solver()
30     s.set(unsat_core=True)
31     for constraint in problem:
32         s.assert_and_track(constraint,str(constraint))
33 
34     if s.check() == sat: # SAT
35         m = s.model()
36         r = []
37         for x in m: # x is a Z3 declaration
38             if is_true(m[x]):
39                 r.append(x()) # x() returns the Z3 expression
40         print(r)
41     else: # unSAT
42         print(s.unsat_core()) # minimal unsatisfiable core
43 
44 system, mcu, network, security, power, battery = Bools('system
    mcu network security power battery')
45 power_budget, total_power = Ints('power_budget total_power')
46 validation_arg = []
47 validation_arg.append(If(battery==True, power_budget <= 250,
    True))
48 validation_arg.append(If(battery==False, power_budget > 250,
    True))
49 validation_arg.append(power_budget >= total_power)
50 validation_arg.append(total_power == (dict['wireless']['wifi']['
    power'] + dict['wireless']['ble']['power']))
51 validation_arg.append(battery)
52 audio = True
53 candidates = []
54 if audio == True:
55     for i in dict['mcu']:
56         for j in dict['mcu'][i]['interface']:
57             if dict['functionality']['audio']['interface'] == j:
58                 candidates.append((i, dict['mcu'][i]['power']))
59 print(candidates[0][0])
60 check(*validation_arg, total_power == candidates[0][1])
61 print("\n\n" + candidates[1][0])
62 check(*validation_arg, total_power == candidates[1][1])
```

Both the power budget and functionality verification are evaluated with the Z3 solver, where a system requirement with audio functionality and battery operated below 250mA is defined. The code from Line 47 to 51 in Code 3.1 shows the definition of the battery power budget constraint of 250mA (Line 47 to 48), definition of total power constraint with respect to power budget (Line 49), definition of total power (Line 50), use of battery as boolean (Line 51), and definition of the requirement of audio functionality. The Z3 solver is then used to verify the available microcontrollers in the JSON

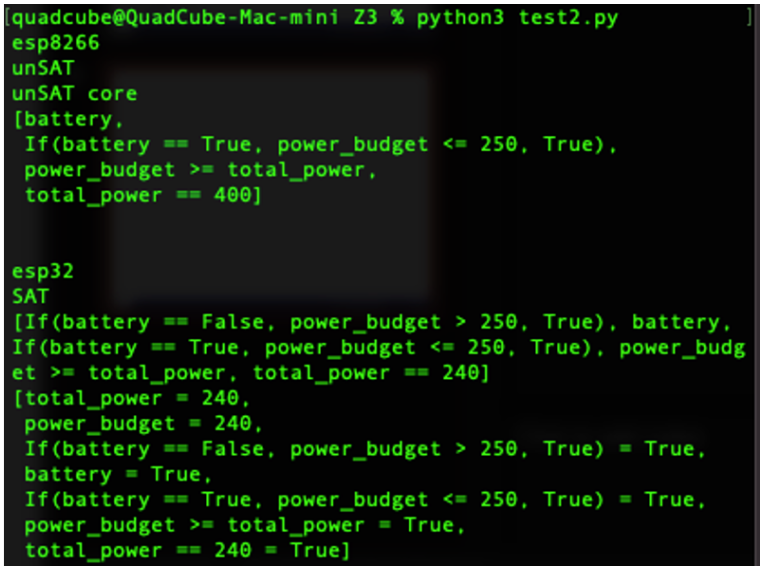

Figure 3.4: Result of Z3 verification for an audio functionality and battery use scenario.

dictionary, where the output result of the verification is shown in Fig. 3.4. Both ESP8266 and ESP32 MCU are evaluated, where the ESP8266 is shown to be unsatisfiable due to it exceeding the power budget and ESP32 is shown to be a satisfiable solution. ESP32 MCU satisfied both power budget and the audio functionality requirement as ESP32 has I2S interface.

Thus, the system specification verification with SMT solver is shown to be viable as proof-of-concept.

3.3.2 System Design Ontology

The previous preliminary work on system specification verification with SMT solver utilized a basic dictionary that describes a few MCUs and their capablities, which is too simplistic for actual IoT system verification. This subsection explores the idea of further expanding such an IoT "dictionary", where more characteristics, features and properties are defined, along with their interlink between the other relevant properties. Ontology is proposed as the approach to encapsulate such information.

An ontology is a representation of types of entities in a given domain and of the relations between them, where it enables structured yet flexible knowledge representation to address structural complexity of complex databases and semantic relationships between the data stored. Besides, it promotes interoperability across heterogeneous subsystems for system design

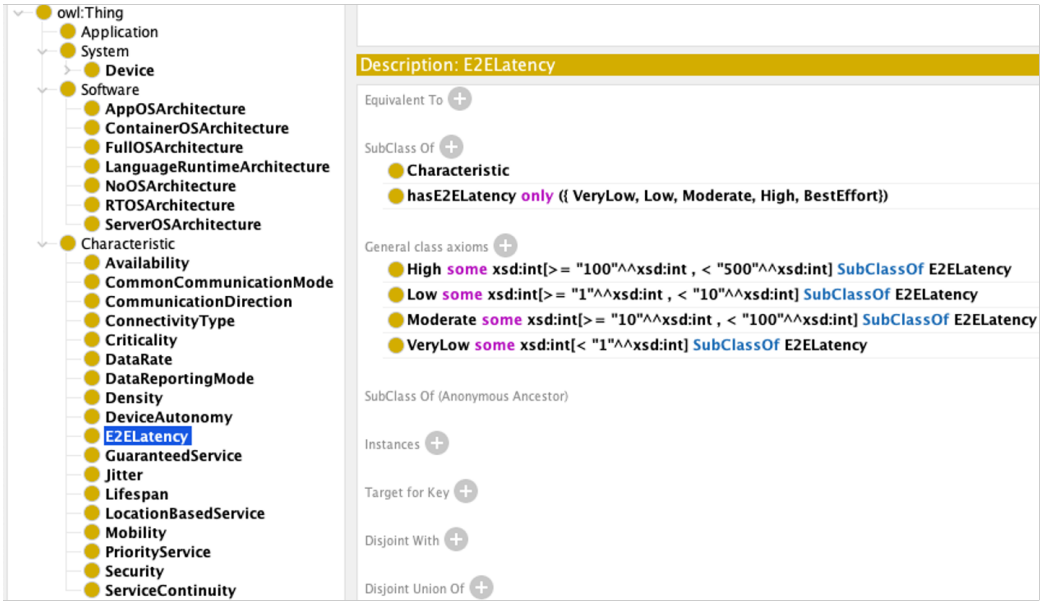

Figure 3.5: Screenshot showing the class and subclass of the prototype ontology in Protégé.

use case. Existing ontologies in the IoT domain mainly focus on machine-to-machine (M2M) protocol interoperability [70, 71]. No existing ontologies in the IoT have such representation for the purpose of system design or system verification.

The prototype ontology represents the IoT device characteristics, their software and system, as shown in Fig. 3.5. Software architecture classifications such as application OS, container OS, real-time OS (RTOS), and many more are defined, while the general characteristics of an IoT device is extensively described in the ontology. Furthermore, the axioms of each individual characteristics are also described, for example: the end-to-end latency ("E2ELatency") has string values describing a range of latency like very low, low, moderate, high, and best effort. To define numerical range for each of these string values, the ontology general class axiom feature is utilized as general class axiom are widely used in domain constraints. As shown in Fig. 3.5, the "High" latency value has a range between 100ms to less than 500ms, where it is also defined as a subclass of the end-to-end latency characteristic. Besides that, the object property hierarchy is also shown in Fig. 3.6, where each performance characteristics are defined.

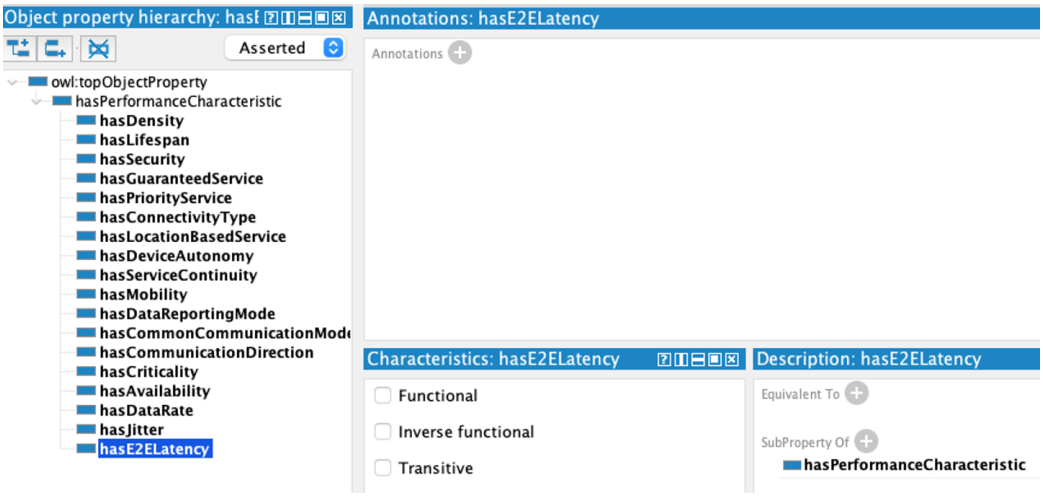

Figure 3.6: Screenshot showing the object property hierarchy of the prototype ontology in Protégé.

3.3.2.1 Integration of SMT Verification with the Ontology

Since the ontology was manually created in the Protégé software, the node transversal in the ontology had to be understood in order to develop automation tools for it. The "Density" characteristic is used as an example, where it has the values of low (< 1000), medium ($1000 <= x < 10000$), high ($>= 10000$), and variable assigned to it. Fig 3.7 illustrates the Resource Description Framework (RDF) triplet for each of the node that are related to the root object "Density" general class axiom, where its following child is linked via the subclass of relationship to the string values of its characteristic. While they are available tools and frameworks to search and parse RDF triplets for common class and subclass objects, no existing framework supports axiom parsing as of the author's knowledge. Hence, a Python tool is developed for RDF general class axiom parsing.

The Python tool is shown in Appendix B, where the RDF general class axiom parser is implemented together with the Z3 input builder. This enables automated encoding of the targeted class/subclass rules and constraints from the linked data in the ontology into Z3 solver programmatically. In order to extract the general class axiom that are linked to the "Density" object, the subclass of "Density" is searched, where the results are the three string values of the axioms (Low, Medium and High). This is done by verifying the following conditions:

- The object ID is equal to the search key

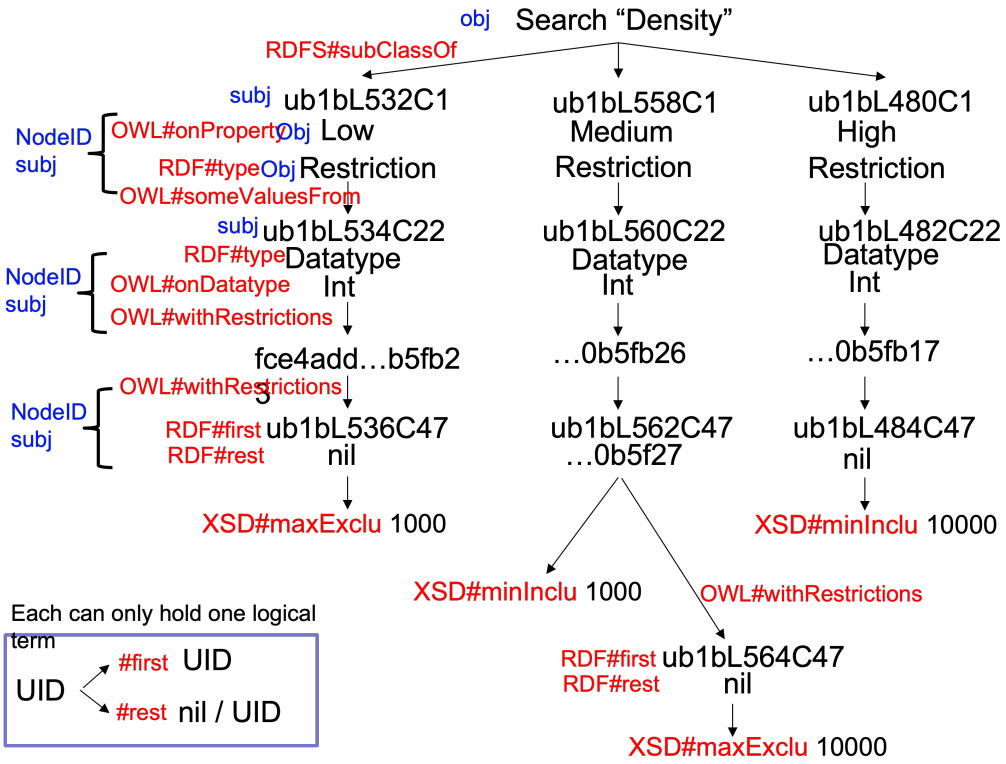

Figure 3.7: An overview of the ontology general class axiom structure in RDF triplet for the “Density” object.

- The predicate is equal to "RDFS.subClassOf"
- The search key type is not equal to "rdflib.term.BNode"
- The subject type is equal to "rdflib.term.BNode"

The resulting triplets includes the individual node ID ("OWL#onProperty") of the string value as the triplet's subject, the corresponding string value as the RDF type ("RDF#type"), and the value "Restriction" for the "OWL#someValuesFrom" subject as shown in Fig 3.7. Hence, the process of searching and parsing the general class axiom is presented.

The second part of the process converts the parsed axioms/constraints of the target subject into individual constraint inputs that are compatible with Z3 solver. Each of the constraints are processed, where its property name (e.g., Low, Moderate, High) is defined into the constraint as a boolean type, and the constraint comparison operators are matched and defined using the following mapping:

- XSD.minInclusive: >=
- XSD.minExclusive: >
- XSD.maxInclusive: <=
- XSD.maxExclusive: <

To show the working of the Python-based RDF general class axiom parser via the Z3 tool, a scenario is evaluated where the end-to-end latency is required to be low in the service requirement and the actual end-to-end latency is measured to be 5ms. Thus, the input conditions of the Python tool are as below:

- Search class: "E2ELatency"
- To be verified level: "Low"
- To be verified actual "E2ELatency" value: 5ms

The verified result is shown in Fig 3.8, where the measured end-to-end latency satisfied the defined low level for "E2ELatency". Thus, the proof-of-concept of integrating the ontology and automating the verification with SMT solver is also successfully conducted.

3.3.3 Analysis of IoT Development and Management Platforms

In this subsection, the IoT platforms presented in Section 2.6 (Bosch IoT Suite, IBM Watson IoT Platform, Microtronics IoT Suite, and IoT Suite) are evaluated as a part of the system specification study during the preliminary work stage of this thesis. This preliminary study provides a comprehensive

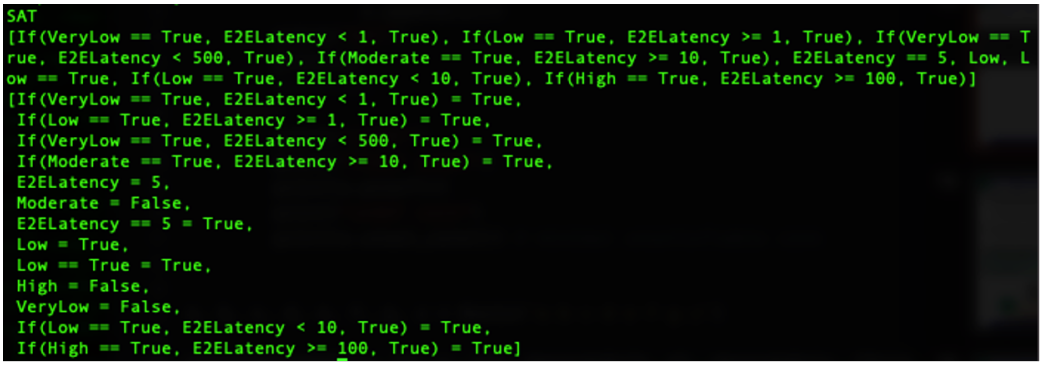

Figure 3.8: Result of automated axiom inclusion from "E2ELatency" into Z3 solver, where the "Low" string value is verified against a latency of 5ms.

insight into the current state-of-art IoT platforms that are available commercially and academic research project.

3.3.3.1 Gap/Overlap Analysis Criteria

To evaluate and compare the IoT platforms, a gap and overlap analysis approach is employed by using reverse mapping on the existing IoT platforms. Each of the IoT platforms are mapped to an IoT reference model, where each of the similar features/characteristics are given an evaluation mapping score from 1 to 3. For the IoT reference model, the Architectural Reference Model (IoT ARM) [3] developed by IoT-A is chosen. The IoT ARM is by far the most comprehensive reference architecture model, as it covers identification and definition of principle features of IoT, universal industry applications examples, and also the use of the IoT reference model with definitions and concepts to analyze and design an IoT architecture. Furthermore, IoT ARM documentations also contains various guidelines and best practices for development. All of these components are illustrated in Fig. 3.9, where each of these building blocks and its interaction with the relevant blocks are shown. Comparing IoT ARM with other state-of-the-art research reference architectures, we conclude that most research paper's reference architectures are written at a high-level view, which typically lack details and depth.

The IoT ARM reference model takes into account various architecture views, such as:

- Physical view
- Context view
- Functional view

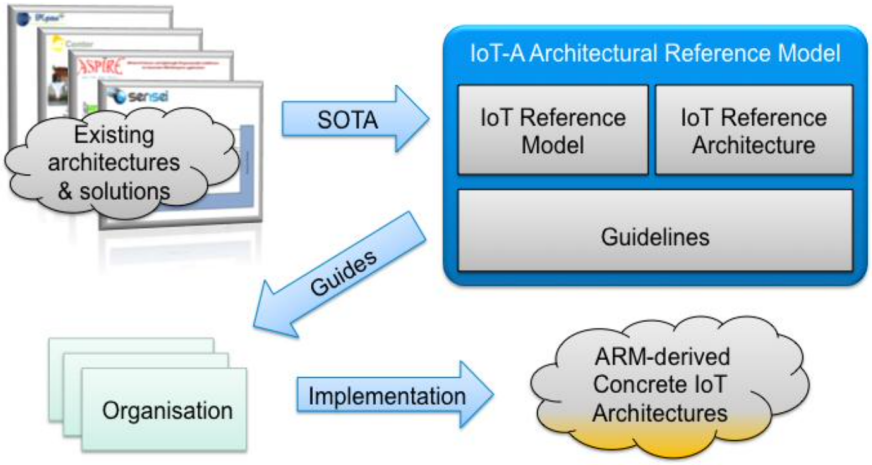

Figure 3.9: IoT ARM building blocks [3].

- Information view
- Deployment view

Besides that, the IoT ARM also covers several architecture perspectives, such as:

- Evolution and interoperability
- Performance and scalability
- Trust, security and privacy
- Availability and resilience

To perform the evaluation of the existing IoT platform, the IoT ARM models are employed as the reference for IoT characteristics/features. The IoT ARM model covers three main models: (i) IoT Domain Model; (ii) IoT Information Model; and (iii) IoT Functional Model.

IoT Domain Model This model is used to capture the main concepts and relationships that are relevant for IoT stakeholders. It covers six key concepts:

- Entities (physical, virtual, augmented)
- Devices
- Resources
- Services
- Identification of Physical Entities
- Context and Location

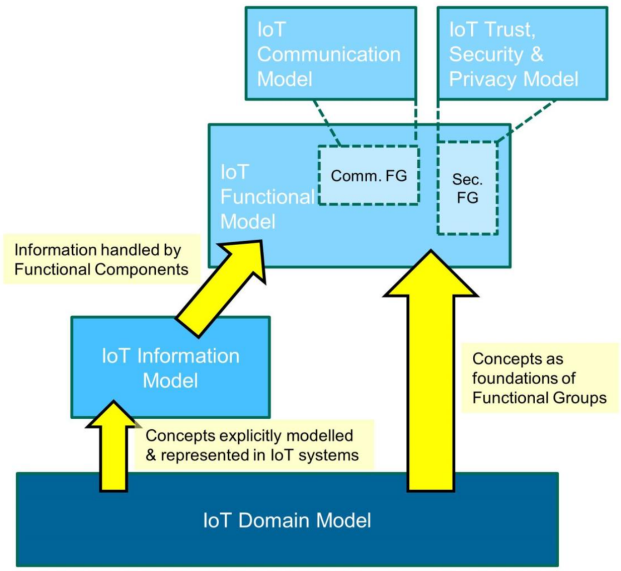

Figure 3.10: IoT ARM domain models and their relationships [3].

IoT Information Model This model defines the structure of all the information for virtual entities on a conceptual level. Examples of such structures are: relations, attributes and services.

IoT Functional Model This model contains both the functionality groups (FG) and the interaction between those parts. There are various functionality groups that are defined under the IoT Functional Model, such as:

- IoT Process Management FG
- Service Organization FG
- Virtual Entity FG
- IoT Service FG
- Communication FG
- Management FG
- Security FG

Each of these models and their relationships are illustrated in Fig. 3.10. The IoT Domain Model is used in the IoT Information Model where its concepts are explicitly modelled and represented in an IoT system, whereas the domain model concepts are also used in the IoT Functional Model as the foundations of the FG.

3.3.3.2 Analysis Results

Using the IoT ARM as the reference model for the gap and overlap analysis, the results of the mapping and its mapping score for each IoT platform are tabulated in Table 3.1 (Bosch IoT Suite), Table 3.2 (IBM Watson IoT Platform), Table 3.3 (Microtronic IoT Suite), and Table 3.4 (IoT Suite).

The Bosch IoT Suite Table 3.1 has a total mapping score 35 out of 42. Out of the total score, 16 out of 18 was scored from the IoT Domain Model, where many of Bosch IoT Suite entities and components are highly similar with the description of the corresponding IoT ARM reference model's component. The Bosch IoT Suite also scored 3 out of 3 in the IoT Information Model, where Bosch's Things feature definition is provided by Eclipse Vorto for device abstraction fully meets the definition of the virtual entity in the IoT Information Model. Lastly, the mapping score in the IoT Functional Model is the lowest, 15 out of 21. The Bosch IoT Suite mapped particularly poorly in areas such as the Service Organization FG while in other areas, such as Communication FG, Management FG and Security FG, the Bosch IoT Suite lacks one or two elements in their corresponding reference definition, which leads to a lower score.

For the IBM Watson IoT Platform, it scores the highest among the four IoT platforms, 38 out of 42. The IBM Watson IoT Platform scored higher than the Bosch in both IoT Domain Model and IoT Functional Model, as the wide range of the IBM cloud services has at least one service that sufficiently maps well to the areas in the reference model. Next, the Microtronic IoT Suite scores a total of 18 out of 42. This low score compared to the previous two platforms are due to the poor or non-existent features/properties in the Microtronic's platform. Note that Microtronic's platform is mainly a rapid prototype environment that does not have a full suite of IoT management capabilities.

The IoT Suite in Table 3.4 has the lowest mapping score out of the four IoT platforms evaluated. One of the main contributing factor to its low mapping score is that the IoT Suite does not have any components that are in the area of IoT Functional Model, as it does not have any management nor any service provided on the platform. Even in areas of the IoT Domain Model, the IoT Suite maps poorly to the corresponding area.

For IoT platforms such as the Bosch IoT Suite and the IBM Watson IoT Platform, it is more intuitive for the user to leverage on its features and services to build an IoT application with end-to-end communication, data analysis, and life-cycle management. The life-cycle management in both of these platforms is done via a simple user interface (UI) and is straightforward to understand. The main difference between these two platforms is that the

Table 3.1: Evaluation of Bosch IoT Suite platform.

IoT ARM	Bosch IoT Suite	Mapping Score
IoT Domain Model		
Entities	Things (physical, virtual, transactional, master data, etc.)	3
Resources	Services (network resources, storage resources)	2
Devices	Things (Things is used to identify both devices and other entities)	2
Services	Services (Connections, Edge downloads, Rollouts, Rules, Tasks, etc.)	3
Identification of Physical Entities	Things ID (namespace)	3
Context and Location	Things Definition and Attributes	3
IoT Information Model		
Virtual Entity	Things feature's definition (device abstraction with Eclipse Vorto)	3
IoT Functional Model		
IoT Process Management FG	Rules, Tasks	2
Service Organization FG	Insight (master data management)	1
Virtual Entity FG	Eclipse Vorto	3
IoT Service FG	Eclipse Vorto, Hub, Things	3
Communication FG	Hub, Things	2
Management FG	Insight, Analytics, Device, Rules, Tasks	2
Security FG	Suite Auth (Authentication, Authorization), Teams, Device Management	2
Total		34

Table 3.2: Evaluation of IBM Watson IoT Platform.

IoT ARM	IBM Watson IoT Platform	Mapping Score
IoT Domain Model		
Entities	Devices (physical, virtual, logical, etc.)	3
Resources	Services (network resources, storage resources, device simulator)	3
Devices	Devices (physical entity)	2
Services	Cloud Services (Bosch IoT Rollout, etc.)	3
Identification of Physical Entities	Device ID	3
Context and Location	Device Information	3
IoT Information Model		
Virtual Entity	Device Information (resource model, event model), Analytics Service (logical interfaces)	3
IoT Functional Model		
IoT Process Management FG	IoT functions	3
Service Organization FG	Cloud Services	3
Virtual Entity FG	Analytics Service (logical interfaces)	2
IoT Service FG	Analytics Service (logical interfaces), Cloud Services, IoT functions	3
Communication FG	Platform Service	2
Management FG	Access Management, Analytic Service (predictions)	3
Security FG	Security, Access Management	2
Total		38

Table 3.3: Evaluation of Microtronic IoT Suite platform.

IoT ARM	Microtronic IoT Suite	Mapping Score
IoT Domain Model		
Entities	Device (physical entity only)	1
Resources	IDE, on-device, network and storage resources	3
Devices	Device, rapidM2M Portal (Sites, Application)	1
Services	rapidM2M Portal (Reports)	2
Identification of Physical Entities	Device serial number	3
Context and Location	Device (except location)	2
IoT Information Model		
Virtual Entity	rapidM2M Portal (Sites, Application)	1
IoT Functional Model		
IoT Process Management FG	-	0
Service Organization FG	-	0
Virtual Entity FG	rapidM2M Portal (Sites, Application, Reports)	1
IoT Service FG	rapidM2M Portal (Sites, Application, Reports)	1
Communication FG	REST API	1
Management FG	rapidM2M Portal (Sites, Application, Reports)	1
Security FG	rapidM2M Portal (Users)	1
Total		18

Table 3.4: Evaluation of IoT Suite platform.

IoT ARM	IoT Suite	Mapping Score
IoT Domain Model		
Entities	Domain specification (vocab.mydsl)	1
Resources	On-device	1
Devices	Vocab.mydsl (physical entity)	1
Services	-	0
Identification of Physical Entities	Deployment specification (deploy.mydsl)	1
Context and Location	deploy.mydsl, vocab.mydsl	3
IoT Information Model		
Virtual Entity	Functional specification (arch.mydsl), deploy.mydsl, vocab.mydsl	3
IoT Functional Model		
IoT Process Management FG	-	0
Service Organization FG	-	0
Virtual Entity FG	-	0
IoT Service FG	-	0
Communication FG	-	0
Management FG	-	0
Security FG	-	0
Total		10

Bosch one lacks integrated virtualization and an analysis service to evaluate the ingested IoT data or device characteristics.

On the other hand, in the IoT Information Model, the solution provided by the Bosch platform, Eclipse Vorto is superior as it provides a descriptive language for definition of IoT devices or services. IoT device that are described in Eclipse Vorto are able to be automatically simulated/emulated and even deployed in both real world and digital twins. Both Eclipse Vorto and the academic project, IoT Suite are useful for the integration and expansion part in automated trustworthiness verification as they are semantically described by design.

Both the Microtronic IoT Suite and the academic project IoT Suite are focused on IoT device development prototyping rather than life-cycle management. Both lack scalability from their prototyping stage or rather they lack a path to scale up. Furthermore, platforms such as Microtronic require proprietary hardware to use their service while IoT Suite is only available on Andriod based device for IoT service implementation.

When it comes to evaluating the platforms from the perspective of security metrics such as trustworthiness validation/verification score, neither of the evaluated platforms do not have such feature. Security level or risk score can be introduced into the IoT Information Model areas, such as devices or network resources. With defined metric score in place, approach such as risk analysis and formal method such as Satisfiability Modulo Theories (SMT) can be utilized to compute the security of the target IoT application.

3.4 Summary

The philosophy of secure system design of this thesis was introduced in this chapter. The topic on trustworthiness is first discussed, followed by the discussion of the characteristics of trustworthiness. The methodology of automated secure system design was then presented, where its requirements were introduced. Furthermore, the preliminary work on automated secure design on IoT, the formal verification of the system specification using Satisfiability Modulo Theories (SMT), ontology based on IoT system design, and the system specification study of existing IoT platforms are presented.

The next chapters in the thesis are as follows. First, the secure design database is presented in Chapter 4. Then, the automated secure system designer verification mechanism and verification functions are presented in Chapter 5. This is followed by case studies of secure system implementation, which are presented in Chapter 6. Finally, the evaluation of the methodology is presented in Chapter 7.

Chapter 4

Secure Design Database

In this chapter, the integral parts that build up the secure design database are introduced. First, the security threat information that is applicable to the system design and its refinement rules is discussed. This is followed by the security knowledge that is build from the MITRE ATT&CK Enterprise matrix that pertains to secure system design. Next, the chapter discusses the approach of extending the security knowledge base using an ontology. Finally, the threat mitigation ontology that is based on the MITRE ATT&CK Enterprise Network domain matrix is presented. The contents of this chapter is a part from the publications [72, 73].

4.1 Secure Design Threats and Rules

In this section, the security threats that are used in secure system design are defined. The threat concept is introduced using the existing Weaver system designer entities, where the concept of logical and conceptual connections are also introduced to extend the capabilities of existing Weaver. The security specific refinement rules are also presented with examples the various types of refinement rules and their corresponding use-case scenarios to illustrate the mechanism of threat mitigation.

4.1.1 Security Threats

In Section 2.4, the entities and intent/service requirement of the existing Weaver were described. Using these concepts, two types of security threats can be defined:

- Component-type threats which are applicable to system components, for example: a physical host
- Relationship-type threats which pertain to relationships between components-type entity, for example: the network connection between a device and a user

Figure 4.1: Service requirement with relationship-type and component-type threats.

For component-type threats, the threat indicates that protection is required at the target component. For relationship-type threats, the entity that requires protection is the attack path to a target component.

An example of a service requirement that illustrates an explicitly-defined relationship-type threat (`Threat1`) applied to the abstract `connTo` relationship between the user `usr`, and the web application `wa`, and a component-type threat (`Threat2`) applied to `wa` is shown in Fig. 4.1. The relationship threat (`Threat1`) in this case denotes that the relationship (`connTo`) is susceptible to eavesdropping or sniffing that may affect the security and privacy of the two end components. On the other hand, the component threat (`Threat2`) denotes that the component (`wa`) is susceptible to a certain malicious activity. Besides, the threat in this thesis examples are represented by an Anonymous mask icon placed on the affected relationship or component entity.

The modeling of a component-type threats is relatively uncomplicated, because this type of threat only affects the target component and its subcomponents that are derived from it during the automated design process. A relationship-type threat however affects multiple components and subcomponents in a system topology. Hence, in order to model the relationship-type threats and their inheritance for affected branches in a topology, the fundamental relationship between two components (e.g., `connTo` relationship between `usr` and `wa` in Fig. 4.1) has to be preserved during the topology refinement process. The objective is to make it possible to verify all possible mitigations for selected topology. For this purpose, the existing Weaver functionality was extended to accommodate such information in the topology refinement process, which is achieved by introducing two new types of connections: (i) logical; and (ii) conceptual.

4.1.2 Logical and Conceptual Connections

Logical connections in this thesis are defined as relationships with an relationship type (*rtype*) property that corresponds application and network layer protocols in the TCP/IP model. Values of the logical connections *rtype* are

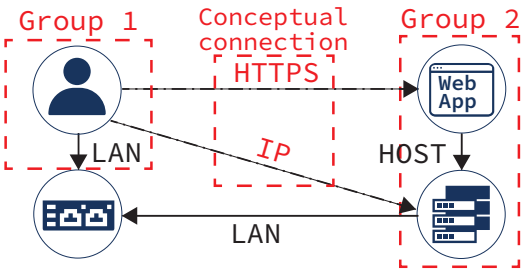

Figure 4.2: Logical and conceptual connections between two system component groups.

concrete protocols in the TCP/IP model, such as `HTTP`, `HTTPS`, `RTP`, `SRTP`, `IPSEC` and `IP`. This is illustrated in Fig. 4.2, where two of such concrete logical connections, `HTTP` and `IPSEC` connects between `Group 1` and `Group 2`, which are defined as dash-dotted lines. The TCP/IP model is found to be well suited for describing the relationship between two network components. Considering only the application and network layers in the TCP/IP model is sufficient for our system design purposes as of writing.

A conceptual connection can be defined as a pair of application and network layer *rtype* entities that connect two component groups in a topology. A component group is denoted as a set of one or more components that were derived via the refinement process from a given component in the service requirement/intent, and that corresponds logically to a service/resource specified in the requirement. This is illustrated in Fig. 4.2, where two such groups are shown: (i) Group 1 made of the user, which forms a group by itself; and (ii) Group 2 made of the web application and the host/machine it is deployed on, as the machine was added to the topology via a refinement rule corresponding to hosting of the web application. The conceptual connection between these two groups is defined as the pair of `HTTPS` and `IP` logical connections. By retaining such logical and conceptual connections in the topology, this preserves the information required by the subsequent security verification algorithm (see Section 5.2.6).

4.1.3 Refinement Rules

In the secure design database, the refinement rules refer to: (i) component refinement; and (ii) relationship refinement. These secure design refinement rules are in addition to the general refinement rules for both components and relationships that are required to design non-security related aspects of the system.

Fig 4.3 shows an overview of security component related refinement

rules. Refinement rules like `DEPLOY-NIDS` and `DEPLOY-FIREWALL` represents the type of refinement rules that add network security appliances into a topology. Furthermore, there are also refinement rules that refine an abstract component into a concrete type for security purposes, for example, the refinement rule `REFINE-VM` refines an abstract `Machine` type component into a `VirtualMachine` instead of the regular `PhysicalMachine` or `PhysicalServer`, which provides isolation or a sandboxed environment.

Next, the refinement rules for the refinement of the relationship type (*rtype*) are explored. Typical refinement rules in the existing Weaver transform a topology state into another by changing or removing the abstract *rtype* or *ctype* entities to achieve a concrete state. On the contrary, for the concrete logical connections that were introduced in the previous subsection, the refinement rules have to be designed in a manner that ensures all the logical connections are preserved in the topology.

One thing to take note of when it comes to security concerns in refinement rules is that top-level threats are explicitly included in the service requirement/intent, but threats associated with protocol-related threats and mitigations are implicitly included via the inherent security properties of TCP/IP model application and network layer protocols when used as an *rtype* in a topology. As a consequent, these are defined in the secure design database's threat mitigation knowledge base, and used for security verification purposes.

Refinement Mechanism For Relationship-Based Threat Mitigations When performing a topology refinement, there may be more than one solution to mitigate a threat. This is illustrated in Fig. 4.4, where a service requirement at topology t_0 has a `User` and a `WebApp` component are connected by the abstract relationship `connTo` to which the threat `T1040` is applied. The abstract relationship `connTo` can be concretized via the usage of either the `HTTP` or `HTTPS` application protocols at the TCP/IP application layer in a conceptual connection, where `HTTP` is insecure and `HTTPS` is secure. The sequential topology refinements for the `HTTP` and `HTTPS` branches are shown in the left and right-hand side diagrams in Fig. 4.4, respectively.

In the left-hand side sequence, the HTTP application protocol does not mitigate the threat `T1040`, therefore the white Anonymous mask icon denotes the associated implicit threat on the affected relationship. Following the subsequent refinement procedure of this topology, the implicit threat is inherited by the `connTo` abstract *rtype* relationship as shown at step $t_{i+n+1,1}$. The term inheritance in this context defines that if the application layer relationship has an implicit threat, the aforesaid threat is also applied implicitly to the following network layer relationship. At step $t_{i+n+2,1}$, the

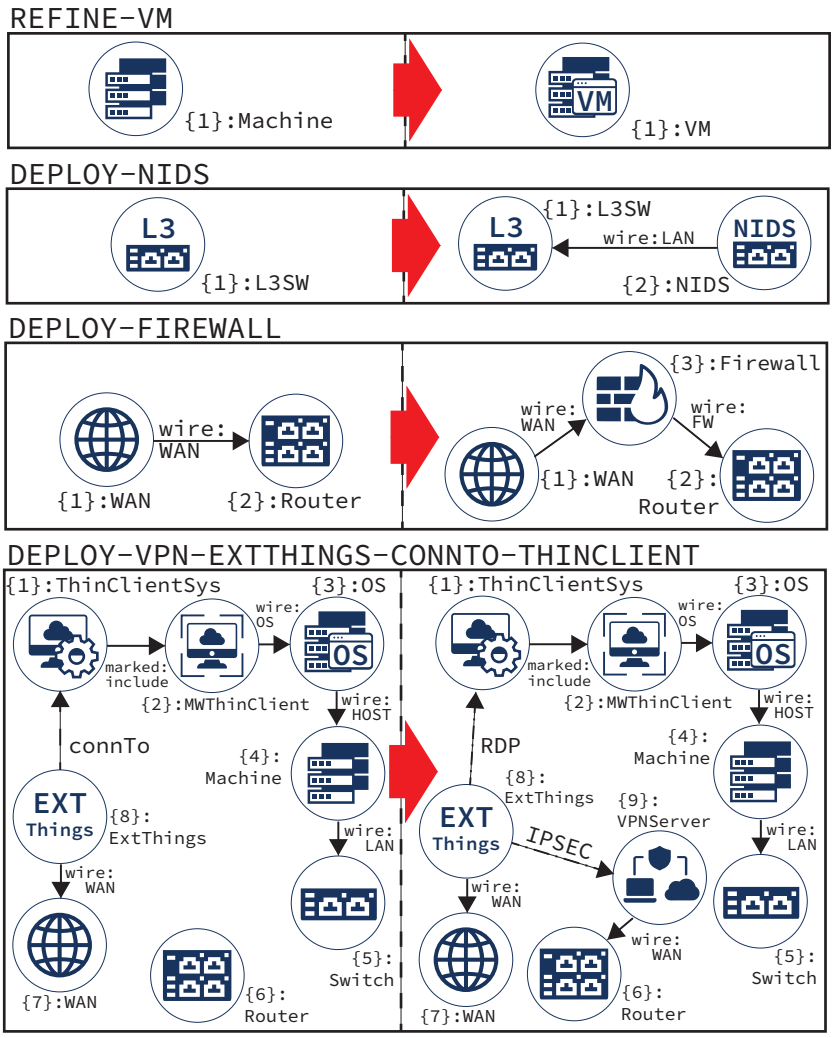

Figure 4.3: Refined web system example scenario without security verification.

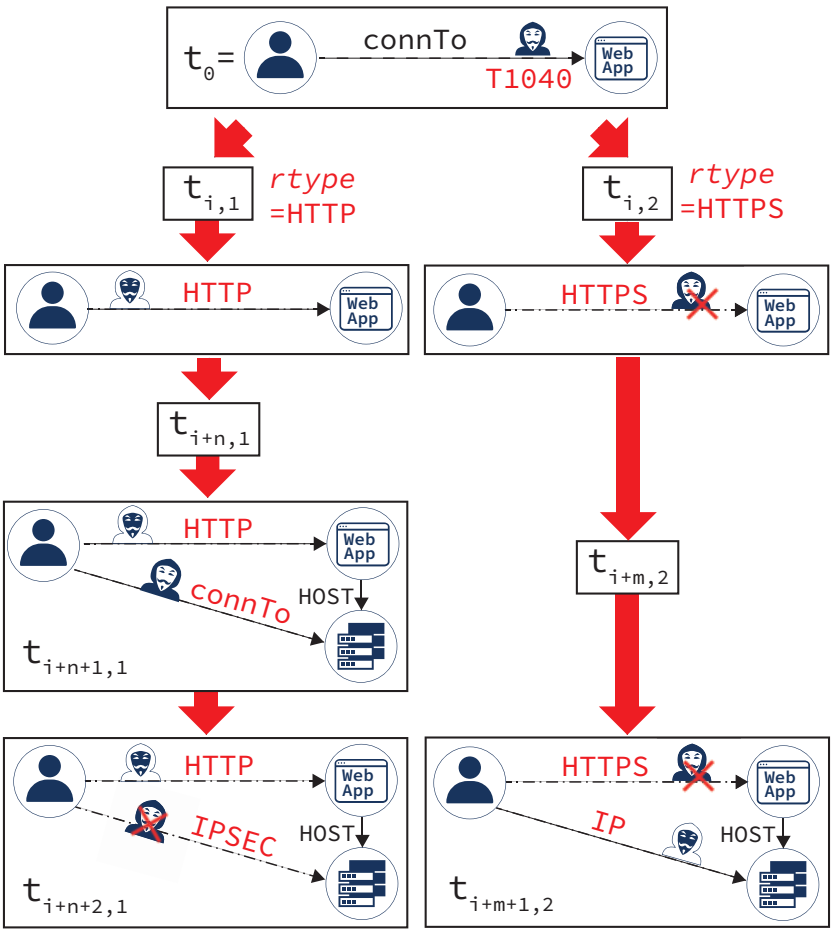

Figure 4.4: Example of possible topology refinements that mitigate the security threat defined in the service requirement t_0.

abstract `connTo` relationship is replaced with a concrete logical connection, `IPSEC`. At this step, the `T1040` threat is mitigated because IPSEC network protocol is defined as a valid mitigation for `T1040` threat in the threat mitigation knowledge base. Threats that are mitigated are denoted with a dark Anonymous mask icon marked with a red "X".

In the right-hand side topology, `HTTPS` directly mitigates the `T1040` threat, as per its secure protocol definition in the threat mitigation knowledge base. Thus, the remaining refinement for concretizing this topology does not have any security issues related to `T1040` such that insecure protocol such as `IP` can be used at network level without compromising the overall security characteristics of the solution.

The discussions about the left-hand side topologies demonstrates that logical connections (insecure) that do not mitigate a threat should not be eliminated immediately by the automated system designer. Doing such would truncate the potential viable solutions. Hence, when it comes to Weaver based system designer, the security verification should only be performed after all the entities in the topology are fully concretized.

4.2 Threat Mitigation Knowledge Base

There are different types of methodologies for security assessment, for instance, the Lockheed Martin Cyber Kill Chain (CKC) [74], MITRE ATT&CK matrix [75], and Microsoft STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege) [76]. The CKC is a well-known intrusion-centric framework which defines a well-established sequence of attack steps. Meanwhile, MITRE ATT&CK matrix is a list of attack techniques that is grouped by tactics, where no specific order of operation are implied in the matrix. The STRIDE framework is a high-level threat model which is typically used during the security development life-cycle since it is focused on identifying overall categories of attacks.

The comparison between security knowledge bases are summarized in Table 4.1, where CKC, ATT&CK and STRIDE are compared in addition to NIST SP 800-53 guidelines and MITRE CVE (Common Vulnerabilities and Exposures). When looking from an abstraction level perspective, its level of detail and orientation, both CKC and STRIDE have very high level of attack model abstraction and vague details as they commonly used to model the possible threat/attack of a system from a high level point-of-view. Guidelines such as NIST SP 800-53 provides a high abstraction level for system defense while at the same time provide very detailed information about its guidelines, while security vulnerability databases such as CVE are

Table 4.1: Comparison between security knowledge bases.

Name	Abstraction level	Level of detail	Orientation	Data format
Lockheed Martin Cyber Kill Chain	Very high	Vague	Attack	Text document
Microsoft STRIDE	Very high	Vague	Attack	Text document
NIST SP 800-53	High	Very detailed	Defense	Text document
MITRE ATT&CK	Medium	Detailed	Attack, Defense	STIX (JSON)
MITRE CVE	Low	Very detailed	Attack	JSON

at lowest abstraction level within the list with very detailed information on the security vulnerabilities is utilized for attack. The ATT&CK matrix, on the other hand, has its abstraction level in the middle, as it describes both attack and defense methods in the respective target domains (Enterprise, Mobile, and Industrial Control Systems (ICS)) in detail. On the data format which the security knowledge base are presented, CKC, STRIDE and SP 800-53 are text documents which rely on user's interpretation of the threat modeling and guidelines for the intended system. On the other hand, both ATT&CK and CVE are available in machine readable JSON format, which can be parsed and implemented in a relatively straightforward way.

Since security threats are explicitly defined in the service requirement (intent) in this thesis's automated system designer that is based off the existing Weaver, the kill chain and high-level modelling techniques are not suitable because they are incompatible with the refinement process in Weaver. In addition, they also do not include sufficient security context. Hence, for threat mitigation knowledge base in the secure design database, the MITRE ATT&CK framework was selected as it provides a rich taxonomy of adversarial tactics, techniques, and common knowledge that can be readily applied in various scenarios.

It is to be noted that other third-party security databases besides MITRE ATT&CK should be compatible for use for security verification in tandem with the MITRE ATT&CK matrix in the threat and mitigation knowledge base. This can be achieved by using common cyber threat intelligence (CTI) sharing language such as Structured Threat Information Expression (STIX) [77], which is discussed in the later section in this chapter (see Chapter 4.3).

4.2.1 MITRE ATT&CK-Based Threat Mitigation

The MITRE ATT&CK based knowledge base is organized as a collection of matrices, such as Enterprise, Mobile and ICS, where each matrix covers a specific field. In a MITRE ATT&CK matrix, the data is categorised into Tactics, Techniques and its sub-techniques, Data Sources, Mitigations, Groups and Software. For the threat mitigation knowledge base in this thesis, the most essential part are the threats and the mitigations. Thus, the MITRE ATT&CK techniques and mitigations are designated as the security threats and mitigations in the threat and mitigation knowledge base.

The version of the MITRE ATT&CK framework that is employed in the threat mitigation knowledge base is version 10.0, which is the latest at the time of writing. Furthermore, the threat mitigation knowledge base specifically focus on the ATT&CK Enterprise matrix, which contains a total number of 14 tactics, 185 techniques, 367 sub-techniques and 42 mitigations.

As the total number of techniques and sub-techniques in ATT&CK is large, the concepts of domain, which is utilized to group relevant techniques and its sub-techniques based on their area of relevance, is leveraged. There are a total number of seven domains in the ATT&CK Enterprise matrix version 10, which are listed below:

- Preparatory
- Windows
- macOS
- Linux
- Cloud
- Network
- Containers

This approach would help cut down the initial number of techniques to a manageable number, which is most relevant to the area of focus in this thesis that is networked system.

The most relevant domain in MITRE ATT&CK Enterprise matrix is the Network domain, as the thesis is focus on networked systems. The network domain has a total number of 9 tactics, 15 techniques, 26 mitigations and 16 data sources, where some of them are unique and some are commonly shared with other domains in the MITRE ATT&CK Enterprise matrix. Each of the threats, mitigations and the affected Weaver entity types for the network domain are tabulated in Table 4.2, where the threats are listed in the order as displayed on the MITRE ATT&CK Web page (https://attack.mitre.org). Some techniques such as `T1600`, `T1056` have no known mitigation in the ATT&CK Enterprise matrix. This is denoted by "N/A" in the Table 4.2.

After analyzing the attack surface/vector for each threat in the table, the affected Weaver entity types (component/relationship) were assigned to their corresponding threat. The description of the mitigation IDs in Table 4.2 is listed in Table 4.3.

By analyzing the ATT&CK network domain's techniques and mitigations, the necessary information required to implement security verification in a system topology for each group of mitigations is obtained. Through this analysis, seven types of verifiable characteristics were identified, as listed below:

1. Application isolation and sandboxing
2. Firewall use
3. Network segmentation
4. Configuration settings
5. Traffic filtering via a network appliance
6. Secure protocol use
7. Intrusion prevention/detection system use

This information is associated to each mitigation and shown in the bottom notes in Table 4.2. Detailed information about the actual verification functions that provide full coverage for the ATT&CK network domain will be presented in Section 5.2.

4.2.2 Structure of the Knowledge Base

For the data structure of the threat mitigation knowledge base, a JSON key-value pair representation was implemented to store the relevant information. The threat mitigation data is organized in three main categories:

- Weight of the security mitigation
- Mitigation verification function mapping
- Secure protocol corresponding to the threat and mitigation pair (if any)

To illustrate the concept of mitigation weight that is introduced, the MITRE ATT&CK Exploit Public-Facing Application (`T1190`) threat is used as an example. The threat `T1190` has six possible mitigations, such as sandboxing mitigation, firewall mitigation, network segmentation, and three mitigations that are related to configuration settings, as shown in Table 4.2. Each of these mitigations may help achieve a particular level of security against a specific target threat. In this thesis, it is assumed that if all available mitigations are applied, the threat is fully negated. Thus, each mitigation can be assigned a different weight, depending on its significance/effectiveness in mitigating/negating a specific threat where the total sum of the weights

Table 4.2: MITRE ATT&CK threats and mitigations in network domain.

Threat	**Mitigations**	**Type**
T1190 (Exploit Public-Facing Application)	M1048[1] M1050[2] M1030[3] M1026[4] M1051[4] M1016[4]	Component
T1059 (Command and Scripting Interpreter)	M1049[4] M1040[4] M1045[4] M1042[4] M1038[4] M1026[4] M1021[4]	Component
T1556 (Modify Authentication Process)	M1032[4] M1028[4] M1026[4] M1025[4] M1022[4]	Component
T1542 (Pre-OS Boot)	M1046[4] M1026[4] M1051[4]	Component
T1205 (Traffic Signaling)	M1042[4] M1037[5]	Component
T1562 (Impair Defenses)	M1022[4] M1024[4] M1018[4]	Component
T1601 (Modify System Image)	M1046[4] M1045[4] M1043[4] M1032[4] M1027[4] M1026[4]	Component
T1599 (Network Boundary Bridging)	M1043[4] M1037[5] M1032[4] M1027[4] M1026[4]	Component
T1600 (Weaken Encryption)	N/A	Component
T1056 (Input Capture)	N/A	Component
T1040 (Network Sniffing)	M1041[6] M1032[4]	Relationship
T1602 (Data from Configuration Repository)	M1041[6] M1037[5] M1031[7] M1030[3] M1054[4] M1051[4]	Component
T1095 (Non-application Layer protocol)	M1037[5] M1031[7] M1030[3]	Component
T1090 (Proxy)	M1037[5], M1031[7] M1020[4]	Component
T1020 (Automated Exfiltration)	N/A	Component

[1] Application isolation and sandboxing
[2] Firewall use
[3] Network segmentation
[4] Configuration settings
[5] Traffic filtering
[6] Secure protocol use
[7] Intrusion detection and prevention system use

Table 4.3: Mitigations applicable to network domain threats.

ID	Name
M1048	Application Isolation and Sandboxing
M1040	Behavior Prevention on Endpoint
M1046	Boot Integrity
M1045	Code Signing
M1043	Credential Access Protection
M1042	Disable or Remove Feature or Program
M1041	Encrypt Sensitive Information
M1038	Execution Prevention
M1050	Exploit Protection
M1037	Filter Network Traffic
M1032	Multi-factor Authentication
M1031	Network Intrusion Prevention
M1030	Network Segmentation
M1028	Operating System Configuration
M1027	Password Policies
M1026	Privileged Account Management
M1025	Privileged Process Integrity
M1022	Restrict File and Directory Permissions
M1024	Restrict Registry Permissions
M1021	Restrict Web-Based Content
M1054	Software Configuration
M1020	SSL/TLS Inspection
M1051	Update Software
M1018	User Account Management
M1016	Vulnerability Scanning

for all the mitigations corresponding to a specific threat is equal to 1.0. The mitigation weight in the threat mitigation knowledge base are by default given an equally distributed weight for the total number of mitigation applicable to a specific threat. It is also envisioned that expert knowledge can be used to assign a more realistic value to the weight each mitigation holds.

Each threat and mitigation pair in the threat mitigation knowledge base is mapped to the corresponding security verification function that is introduced in Section 4.2.1. The pairs are also tagged with their relevant Weaver entity type (component/relationship), which denotes the type of entity the mitigation can be applied to. While the parameters are similar across most type of mitigations in the knowledge base, the configuration setting based mitigations have an extra parameter that denotes whether the corresponding system setting are assumed to be applied or not, which is simply designated by `True` or `False` as its value. As of writing, Weaver system designer does not support any functionality to precisely model an entity with detailed software/configuration settings. Hence, if Weaver eventually gains such functionality to verify the actual settings, the configuration setting could be dynamically verified as well.

The threat mitigation knowledge base also includes the information about the logical connections introduced in Section 4.1.2. The logical connections that are secure network protocols such as `HTTPS`, `IPSEC` and `SRTP` are defined in the knowledge base and mapped to the corresponding threat and mitigation pair. Hence, any logical connection relationship types that are not explicitly denoted in the threat mitigation knowledge base are considered to be insecure.

4.3 Ontology Extension of the Threat Mitigation Knowledge Base

To improve the quality of the threat mitigation knowledge base in the secure design database, one should refer to more than one trusted third-party data source. Some examples of trusted third-party data source from MITRE and NIST are the ATT&CK matrix [78], Common Attack Pattern Enumeration and Classification (CAPEC) [79], Common Vulnerability Enumerations (CVE) [80], Common Weakness Enumerations (CWE) [81], and National Vulnerability Database (NVD) [82]. Referencing multiple trusted data sources generally improves the trustworthiness of the required information needed for security verification. Besides that, each data source may present

different viewpoints of the same information for a certain subject such as CVE provides concrete information of a vulnerable target, while ATT&CK provides a relatively abstract objective of an Advanced Persistent Threat (APT) on how the vulnerable target could be exploited [28].

Such extension to accommodate various data sources requires the use of semantics to link the information. In this section, an ontology is proposed, as this is a developed field in semantic technologies that provides various languages such as Resource Description Framework (RDF), Web Ontology Language (OWL), and various other tools like reasoners and queries. Semantic relationships are the core behind an ontology, as it maps/links multiple data together to create a shared vocabulary in cyber security.

A Cyber Threat Intelligence (CTI) typically describes detailed information about threats and threat actors in cyber security. Details such as timestamp and location of attack, type of malware, threat actor group, attack vectors, and indicator of compromise (IOC) such as source IP address are usually defined in a CTI. Since CTI sharing is common between government and industries, much works have gone into standardizing the format of CTI for effective and efficient CTI sharing. While this thesis does not go into detail on how to model a CTI, there are various models to describe a CTI and its use, such as the Detection Maturity Level (DML) model [83], Cyber Kill Chain (CKC) [74], and CTI [84]. Moreover, there are also various data formats that are used for CTI exchanges, such as:

- Open Indicators of Compromise (OpenIOC) [85]
- Incident Object Description Exchange Format (IODEF) [86]
- Vocabulary for Event Recording and Information Sharing (VERIS) [87]
- Structured Threat Intelligence Exchange (STIX) [5]

OpenIOC mainly focuses on aiding an investigator to describe and classify artifacts encountered during the source of a security incident investigation. IODEF primary focus is to enable incident information exchanges between Computer Emergency Response Teams (CERTs), while VERIS lies emphasis on cyber security incident's measurement and management of risks. STIX on the other hand is not confined to specific use case while providing an extensive tool set to aid representation of various information about cyber security incidents [88]. Furthermore, the STIX data format is the de-facto standard for CTI used in the industry [89] and its also the format with the most extensive capabilities in application. Thus, STIX is chosen as the structured representation to build the ontology. The next subsection will introduce STIX and its use in MITRE ATT&CK matrix, followed by the ontology based on Network domain in ATT&CK Enterprise matrix.

Table 4.4: Examples of STIX v2.1 domain objects [5].

Name	Type name	Description
Attack pattern	attack-pattern	A type of Tactics, Techniques, and Procedures (TTP) that describe the approaches that adversaries attempt to compromise its targets
Course of action	course-of-action	A recommendation from a producer of intelligence to a consumer on the actions that they might take in response to that intelligence
Identity	identity	Actual individuals, organizations, or groups (e.g., ACME, Inc.) as well as classes of individuals, organizations, systems or groups (e.g., the finance sector)
Intrusion set	intrusion-set	A grouped set of adversarial behaviors and resources with common properties that is believed to be orchestrated by a single organization
Malware	malware	A type of TTP that represents malicious code
Tool	tool	Legitimate software that can be used by threat actors to perform attacks

Structured Threat Intelligence Exchange (STIX) Structured Threat Intelligence Exchange (STIX) is a state-of-the-art semi-structured representation for CTI that is developed by OASIS Open for the purpose of sharing CTI. STIX is graph based model where a piece of information is represented as an object with its attributes, and the information are linked through relationship. Thus, the two core object type in STIX are:

- STIX Domain Object (SDO)
- STIX Relationship Object (SRO)

An SDO describes the characteristics of an incident, while an SRO describes the relationships between those characteristics. The current version of STIX as of writing is version 2.1, where there are a total of 18 SDO and 2 SRO. Table 4.4 lists some of the SDO that are used in the MITRE ATT&CK STIX collection, where the full list of SDO can be referred in [5]. Table 4.5 lists all the SROs that are available in STIX.

Each STIX object (SDO/SRO) has their properties defined using the common properties as shown in Table 4.6. While a STIX object may use all the available common properties, some may just define the required properties as listed in Table 4.6.

Table 4.5: STIX v2.1 relationship objects [5].

Name	Type name	Description
Relationship	relationship	Used to link together two SDOs or SCOs in order to describe how they are related to each other
Sighting	sighting	Denotes the belief that something in CTI (e.g., an indicator, malware, tool, threat actor, etc.) was seen

Table 4.6: Common properties of STIX v2.1 objects [5].

Name	Type	Description	Requirement
type	string	Identifies the type of STIX Object.	Required
spec_version	string	The version of the STIX specification used to represent this object.	Required
id	identifier	Uniquely identifies this object.	Required
created_by_ref	identifier	Specifies the id property of the "identity" object that describes the entity that created this object.	Optional
created	timestamp	Represents the time at which the object was originally created.	Required
modified	timestamp	Only used by STIX Objects that support versioning and represents the time that this particular version of the object was last modified.	Required
revoked	boolean	Only used by STIX Objects that support versioning and indicates whether the object has been revoked.	Optional
labels	list of type "string"	Specifies a set of terms used to describe this object	Optional
confidence	integer	Identifies the confidence that the creator has in the correctness of their data.	Optional
lang	string	Identifies the language of the text content in this object.	Optional

external_references	list of type "external_reference"	Specifies a list of external references which refers to non-STIX information.	Optional
object_marking_refs	list of type "identifier"	Specifies a list of "id" properties of "marking-definition" objects that apply to this object.	Optional
granular_markings	list of type "granular_marking"	Specifies a list of granular markings applied to this object.	Optional
defanged	boolean	Defines whether or not the data contained within the object has been defanged.	N/A
extensions	dictionary	Specifies any extensions of the object, as a dictionary.	N/A

The STIX incident information representation structure is written in the JSON format. An example of a threat report on Poison Ivy trojan from non-STIX source [90] represented in STIX version 2.1, where part of the "attack-pattern" and "course-of-action" are shown in Code 4.1 and Code 4.2, respectively. The threat report in [90] compiles how various threat actors in different campaigns used the Poison Ivy remote access tool (RAT) for malicious activities. The example of an "attack-pattern" representation presented in Code 4.1 shows all the required common properties in Table 4.6 as well as optional properties specific to "attack-pattern" SDO such as "kill_chain_phases".

Code 4.1: Representation of STIX v2.1 "attack-pattern" for Poison Ivy threat report [4].

```
{
    "type": "attack-pattern",
    "spec_version": "2.1",
    "id": "attack-pattern--19da6e1c-69a8-4c2f-886d-d620d09d3b5a",
    "created": "2015-05-15T09:12:16.432Z",
    "modified": "2015-05-15T09:12:16.432Z",
    "external_references": [
      {
        "source_name": "capec",
        "description": "spear phishing",
        "external_id": "CAPEC-163"
      }
    ],
    "name": "Spear Phishing Attack Pattern used by admin@338",
```

```
15      "description": "The preferred attack vector used by
            admin@338 is spear-phishing emails. Using content that is
             relevant to the target, these emails are designed to
            entice the target to open an attachment that contains the
             malicious PIVY server code.",
16      "kill_chain_phases": [
17        {
18          "kill_chain_name": "mandiant-attack-lifecycle-model",
19          "phase_name": "initial-compromise"
20        }
21      ]
22  }
```

The mitigation for the "attack-pattern" is shown in Code 4.2, where "action_reference" points to a valid external reference that a human expert can refer to. For setting commands that can be applied directly to mitigate the threat in a machine, STIX also provides an optional "action_bin" property in replacement of "action_reference" in base64 encoded command for such use case. This example illustrates the flexibility of STIX version 2.1 to accommodate various use cases.

Code 4.2: Representation of STIX v2.1 "course-of-action" for Poison Ivy threat report [4].

```
1   {
2       "type": "course-of-action",
3       "spec_version": "2.1",
4       "id": "course-of-action--8e2e2d2b-17d4-4cbf-938f-98
            ee46b3cd3f",
5       "created_by_ref": "identity--f431f809-377b-45e0-aa1c-6
            a4751cae5ff",
6       "created": "2016-04-06T20:03:48.000Z",
7       "modified": "2016-04-06T20:03:48.000Z",
8       "name": "mitigation-poison-ivy-firewall",
9       "description": "This action points to a recommended set of
            steps to respond to the Poison Ivy malware on a Cisco
            firewall device",
10      "action_type": "cisco:ios",
11      "action_reference":
12          { "source_name": "internet",
13            "url": "https://www.stopthebad.com/poisonivyresponse.
                  asa"
14          }
15  }
```

For the SRO, it labels the relationship(s) between two SDO. An example of the "attack-pattern" SDO relationship to other SDOs in STIX is visualized in Fig. 4.5. Some of the SDOs, such as "campaign", "course-of-action",

Figure 4.5: Visualized STIX "attack-pattern" SDO relationships [4].

"indicator", and many more have their relationship mapped to "attack-pattern". Besides that, there can also be more than one relationship between two SDOs, such as "attack-pattern" and "malware", where it has bi-direction "Uses" that represents both SDOs can use one another to achieve its target and "Delivers" shows that an attack pattern can be used to deliver a specific malware. This provides the semantic linking between different SDOs, which is useful for creating the ontology in the later section.

Fig. 4.6 shows the relationships between "course-of-action" and the SDOs related to it. An example of multiple relationship between two SDOs is also illustrated, where "course-of-action" can both "Remediates" and "Mitigates" the malware and vulnerability type SDOs. In fact, the "Remediates" SRO means that the threat can be eradicated, which also includes "Mitigation" as a mitigation is akin to damage control where the issue cannot be eliminated.

4.4 MITRE ATT&CK-Based Ontology

The entire collection of MITRE ATT&CK matrices are available in STIX version 2.0 [91] and STIX version 2.1 [6] formats. In this thesis, the version 10.0 of MITRE ATT&CK Enterprise matrix in STIX version 2.1

Figure 4.6: Visualized STIX "course-of-action" SDO relationships [4].

is employed to build the ontology. In order to build the basic ontology structure that is similar to the threat mitigation knowledge base introduced in Section 4.2.1, the tactic, technique and mitigation concepts in MITRE ATT&CK are used. First, the MITRE ATT&CK Enterprise matrix in STIX representation is explored. This includes the statistical overview of the entire JSON bundle/collection for MITRE ATT&CK Enterprise matrix version 10.0. Then, the work on mapping/linking the SDOs related to MITRE ATT&CK tactic, technique and mitigation in the JSON bundle is discussed, followed by the result of the MITRE ATT&CK based ontology.

4.4.1 Exploring the MITRE ATT&CK Enterprise Matrix in STIX

While MITRE ATT&CK Enterprise matrix can be represented in STIX, it requires extensions as custom objects to fully represent all the information. An example of a "course-of-action" in STIX version 2.1, ATT&CK Network Segmentation mitigation (M1041) is shown in Code 4.3. The custom properties, such as "x_mitre_version", "x_mitre_attack_spec_version", "x_mitre_domains", "x_mitre_modified_by_ref" in Code 4.3 are all MITRE ATT&CK custom STIX objects.

Code 4.3: JSON example of a MITRE ATT&CK Enterprise in STIX 2.1 "course-of-action" object [6].

```
{
    "id": "course-of-action--86598de0-b347-4928-9eb0-0
        acbfc21908c",
    "name": "Network Segmentation",
    "description": "Architect sections of the network to isolate
         critical systems, functions, or resources. Use physical
        and logical segmentation to prevent access to potentially
         sensitive systems and information. Use a DMZ to contain
        any internet-facing services that should not be exposed
        from the internal network. Configure separate virtual
        private cloud (VPC) instances to isolate critical cloud
        systems.",
    "external_references": [
        {
            "source_name": "mitre-attack",
            "external_id": "M1030",
            "url": "https://attack.mitre.org/mitigations/M1030"
        }
    ],
    "object_marking_refs": [
        "marking-definition--fa42a846-8d90-4e51-bc29-71
            d5b4802168"
    ],
    "created_by_ref": "identity--c78cb6e5-0c4b-4611-8297-
        d1b8b55e40b5",
    "type": "course-of-action",
    "modified": "2020-05-14T13:05:39.500Z",
    "created": "2019-06-10T20:41:03.271Z",
    "x_mitre_version": "1.1",
    "spec_version": "2.1",
    "x_mitre_attack_spec_version": "2.1.0",
    "x_mitre_domains": [
        "enterprise-attack"
    ],
    "x_mitre_modified_by_ref": "identity--c78cb6e5-0c4b
        -4611-8297-d1b8b55e40b5"
}
```

The MITRE ATT&CK Enterprise matrix is stored inside a STIX bundle in a JSON file. Table 4.7 shows the top-level structure of the STIX bundle, where some of STIX common properties are described. The "objects" entry is a list array that stores the entire ATT&CK Enterprise matrix SDOs, SROs and other metadata in a flat structure.

In order to work with the STIX bundle, a short script was developed to parse the STIX bundle and list all the keys and their number of occurrences.

Table 4.7: Top-level structure of MITRE ATT&CK STIX bundle.

Object	Value
type	bundle
id	bundle–{bundle uuid}
spec_version	2.1
objects	[...]

Algorithm 1 describes the overall flow to parse and count every object keys in the "objects" list in the STIX bundle. The algorithm first parse the "x-mitre-collection" key, as this key contains all the object reference, "object_ref" and modified timestamp, "object_modified" information in its internal dictionary structure, "x_mitre_contents". For every type of object that is not found in the "data_dict" array, the information of the object and count is appended into the array. If the type of object is found in the array, the count for that object is incremented by 1.

After the "x-mitre-collection" is parsed and processed, the algorithm starts parsing the large and flat "objects" list. Similar to the "x-mitre-collection" routine, the type of object is entered into the array if they are not found while the existing ones are incremented for all objects and their sub-objects.

The result of running Algorithm 1 is shown in Table 4.8. To verify the statistics in Table 4.8, the total count of an object is compared to the MITRE ATT&CK matrix. Since "course-of-action" is known to be the mitigation in MITRE ATT&CK, the values are compared. The total number of "course-of-action" is 284, wherea MITRE ATT&CK Enterprise matrix only have 43 mitigations. Further investigation into the large disparity reveals that the STIX bundle for MITRE ATT&CK not only contains the version 10.0, it also contains all the older versions of MITRE ATT&CK where their SDOs are marked with "revoked" as True. After accounting for the revoked/deprecated SDOs in the STIX bundle, the valid MITRE ATT&CK concepts can be obtained.

Since Algorithm 1 also parses and processes the "x_mitre_contents" in "x-mitre-collection", those results are tabulated in Table 4.9. The "x_mitre_contents" stores all the UUID of the objects in the STIX bundle, where the results in Table 4.8 and Table 4.9 can be compared and verified.

Table 4.8: Statistical result of MITRE ATT&CK Enterprise matrix in STIX.

Type	Name	No. of entries
STIX Domain Objects (SDO)	attack-pattern	707
	relationship	14467
	course-of-action	284
	identity	1
	intrusion-set	136
	malware	475
	tool	73
	marking-definition	1
ATT&CK Custom Object	x-mitre-collection	1
	x-mitre-tactic	14
	x-mitre-matrix	1
	x-mitre-data-source	38
	x-mitre-data-component	109

Table 4.9: MITRE ATT&CK Enterprise matrix in STIX custom object "x_mitre_contents".

Name	No. of entries
attack-pattern–{uuid}	707
relationship–{uuid}	14467
course-of-action–{uuid}	284
identity–{uuid}	1
intrusion-set–{uuid}	136
malware–{uuid}	475
tool–{uuid}	73
x-mitre-tactic–{uuid}	14
x-mitre-matrix–{uuid}	1
x-mitre-data-source–{uuid}	38
x-mitre-data-component–{uuid}	109

Algorithm 1 Parse and process MITRE ATT&CK Enterprise matrix STIX bundle.

Input: STIX bundle, *bundle*
Output:

```
1: function PARSE_STIX_BUNDLE(bundle)
2:     obj ← bundle["objects"]
3:     for all data in obj["x-mitre-collection"] do
4:         data1 ← data["type"]
5:         if data1 not in data_dict then
6:             data_dict ← [data1, 1]
7:         else
8:             Increment count in data_dict[data1]
9:         end if
10:    end for
11:    for all data in obj["objects"] do
12:        data1 ← data["type"]
13:        if data1 not in data_dict then
14:            data_dict ← [data1, 1]
15:            for all obj_key in data do
16:                obj_dict[data1][obj_key] ← 1
17:            end for
18:        else
19:            Increment count in data_dict[data1]
20:            for all obj_key in data do
21:                if obj_key not in obj_dict[data1] then
22:                    obj_dict[data1][obj_key] ← 1
23:                else
24:                    Increment count in obj_dict[data1][obj_key]
25:                end if
26:            end for
27:        end if
28:    end for
29:    return data_dict, obj_dict
30: end function
```

4.4.2 Rebuilding Semantic Links from MITRE ATT&CK STIX Bundle

The approach to transverse the STIX bundle was discussed in the previous subsection. By extending the previous approach, the semantic link between

Table 4.10: Example of MITRE ATT&CK Enterprise matrix concepts mapping to existing STIX v2.1 domain objects.

MITRE ATT&CK concept		STIX object	
Object	Property	Object	Property
Tactic	name description	attack-pattern	kill-chain-phases name description
Technique	name description tactic	attack-pattern	name description kill-chain-phases
Mitigation	name description	course-of-action	name description

each SDO can be obtain and the graph can be rebuilt in the appropriate format that is compatible with ontology editor and knowledge management system such as Protégé. The ontology that will be built will focus on the following MITRE ATT&CK concepts:

- Tactics
- Techniques
- Mitigations
- Data sources

The concepts are obtained from the network domain in the MITRE ATT&CK Enterprise matrix to populate the resulting ontology. Table 4.10 shows an example of MITRE ATT&CK Enterprise concepts mapping to existing SDOs. While not all concepts can be mapped one-to-one, the mapping shows how STIX is generally utilized to represent MITRE ATT&CK concepts. The one-to-one mapping from MITRE ATT&CK concepts to their respective STIX object type is shown in Table 4.11, where both existing STIX SDOs and custom STIX objects are used. ATT&CK tactics are denoted as "x-mitre-tactic" when used to describe the specific tactic information, while the "kill-chain-phases" property of "attack-pattern" or ATT&CK technique is used to list all the phases in the kill chain.

An example for the MITRE ATT&CK network sniffing threat T1040 in STIX, the "kill-chain-phases" stores two ATT&CK tactic: (i) Credential Access; and (ii) Discovery. In the entry for ATT&CK tactic Credential Access, the "kill-chain-name" and "phase_name" values are "mitre-attack"

Table 4.11: MITRE ATT&CK Enterprise matrix concept mapping to STIX object type in STIX bundle [6].

ATT&CK concept	STIX object type	Custom type
Tactics	x-mitre-tactic	Yes
Techniques	attack-pattern	No
Mitigations	course-of-action	No
Data sources	x-mitre-data-component	Yes

and "credential-access" respectively. The actual semantic link or relationship is describe as SRO in the STIX bundle. Hence, query functions are developed to search for the STIX UUID or ATT&CK concept specific ID and filter out the deprecated SDOs and SROs.

The flow to create the ontology is first to parse and process the raw SDO and SRO data in the MITRE ATT&CK STIX bundle, then to create a JSON-LD representation of the data and their semantic links. The resulting JSON-LD can be directly imported into the Protégé ontology software. First, the JSON-LD structure starts with the base ontology, *onto_base* as shown in Code 4.4. This mainly describes the ontology with the "http://www.w3.org/2002/07/owl#Ontology" type and ATT&CK concepts as general classes via "http://www.w3.org/2000/01/rdf-schema#Class" in the ontology. Next, to populate the ontology with MITRE ATT&CK information, a Python 3 script was developed to parse and process data in the ATT&CK in STIX bundle and append them into the *onto_base*.

Code 4.4: JSON-LD base ontology.

```
1  [
2      {
3          "@id": "https://attack.mitre.org/",
4          "@type": [
5              "http://www.w3.org/2002/07/owl#Ontology"
6          ],
7          "http://www.w3.org/2000/01/rdf-schema#label": "MITRE ATT
               &CK MATRIX",
8          "http://www.w3.org/2000/01/rdf-schema#comment": "Network
               domain collection"
9      },
10     {
11         "@id": "https://attack.mitre.org/techniques",
12         "@type": [
13             "http://www.w3.org/2000/01/rdf-schema#Class"
14         ]
15     },
```

```
16      {
17          "@id": "https://attack.mitre.org/tactics",
18          "@type": [
19              "http://www.w3.org/2000/01/rdf-schema#Class"
20          ]
21      },
22      {
23          "@id": "https://attack.mitre.org/mitigations",
24          "@type": [
25              "http://www.w3.org/2000/01/rdf-schema#Class"
26          ]
27      },
28      {
29          "@id": "https://attack.mitre.org/datasources",
30          "@type": [
31              "http://www.w3.org/2000/01/rdf-schema#Class"
32          ]
33      }
34  ]
```

This is shown in Algorithm 2, where all the techniques in the MITRE ATT&CK Enterprise network domain are retrieved. Deprecated techniques in the network domain are also filtered by *get_techniques_by_platform*(). With the network domain techniques retrieved, each of the technique's tactic (*kill_chain_phases*) is obtained and appended into a temporary variable *tactics*1 as shown from Line 5 to 11.

Next, the mitigation(s) for each technique are obtained via the function *get_mitigations_by_technique_uuid*(), where the UUID of the selected network domain technique is provided. Similar to the previous procedure, the deprecated mitigations are also filtered during retrieval and its result is appended to *tactics*1 as shown from Line 12 to 18. The data source(s) of the selected technique is obtained via the technique UUID and each of the results is appended to *tactics*1. The selected technique together with the ontology data in *tactics*1 are then appended into *onto_base*. This is repetitively done for all the techniques in the network domain in ATT&CK Enterprise matrix. Lastly, the ATT&CK data sources, mitigations and tactics information are appended into the *onto_base* as shown in Line 28 to 39, and the *onto_base* is returned as the JSON-LD result. The Python 3 source code for this algorithm is made available on Github [92].

A short snippet of the resulting JSON-LD output is shown in Code 4.5, where MITRE ATT&CK threat T1040 and its relationships are described. The "@id" denotes the ID for treat T1040 points directly to its URL on MITRE ATT&CK webpage. The "@type" defines both ontology RDF class and MITRE ATT&CK technique as its type. The RDF schema label is

used for defining the name of the threat. Furthermore, the RDF schema "subClassOf" is used to define the semantic links to other SDOs such as ATT&CK technique class, individual tactics, mitigations and data sources, respectively.

Code 4.5: JSON-LD result of MITRE ATT&CK threat T1040 and its respective semantic links.

```
{
    "@id": "https://attack.mitre.org/techniques/T1040",
    "@type": [
        "http://www.w3.org/2000/01/rdf-schema#Class",
        "https://attack.mitre.org/techniques"
    ],
    "http://www.w3.org/2000/01/rdf-schema#label": "Network
        Sniffing",
    "http://www.w3.org/2000/01/rdf-schema#comment": "Adversaries
         may sniff network traffic to capture information about
        an environment, including authentication material passed
        over the network. Network sniffing refers to using the
        network interface on a system to monitor or capture
        information sent over a wired or wireless connection. An
        adversary may place a network interface into promiscuous
        mode to passively access data in transit over the network
        , or use span ports to capture a larger amount of data.\n
        \nData captured via this technique may include user
        credentials, especially those sent over an insecure,
        unencrypted protocol. Techniques for name service
        resolution poisoning, such as [LLMNR/NBT-NS Poisoning and
         SMB Relay](https://attack.mitre.org/techniques/T1557
        /001), can also be used to capture credentials to
        websites, proxies, and internal systems by redirecting
        traffic to an adversary.\n\nNetwork sniffing may also
        reveal configuration details, such as running services,
        version numbers, and other network characteristics (e.g.
        IP addresses, hostnames, VLAN IDs) necessary for
        subsequent Lateral Movement and/or Defense Evasion
        activities.",
    "http://www.w3.org/2000/01/rdf-schema#subClassOf": [
        {
            "@id": "https://attack.mitre.org/techniques"
        },
        {
            "@id": "https://attack.mitre.org/tactics/TA0006"
        },
        {
            "@id": "https://attack.mitre.org/tactics/TA0007"
        },
        {
```

```
20                "@id": "https://attack.mitre.org/mitigations/M1032"
21            },
22            {
23                "@id": "https://attack.mitre.org/mitigations/M1041"
24            },
25            {
26                "@id": "https://attack.mitre.org/datasources/DS0009"
27            },
28            {
29                "@id": "https://attack.mitre.org/datasources/DS0017"
30            }
31        ]
32    }
```

The result of the import of the JSON-LD file into Protégé ontology software is shown in Fig. 4.7, where on the left-hand side under the class hierarchy, the data sources, mitigations, tactics and techniques in MITRE ATT&CK network domain are listed with their respective sub-classes. On the top right of Fig. 4.7, all usage of threat with ID T1040 is shown, which lists all the sub-classes related to T1040, its "rdfs:comment" as the description of the threat and the label or name of T1040.

4.5 Summary

In this chapter, each of the requisite components in the secure design database were presented. The concept behind a security threat for system design was discussed, together with the extension of the existing Weaver relationship concept to accommodate the relationship type security threats. This was followed by the discussion on various third-party security knowledge base, where Network domain of MITRE ATT&CK Enterprise matrix was chosen for the threat mitigation knowledge base. The mitigations in the network domain were also analyzed which results in isolating seven types of verifiable characteristics. Furthermore, the chapter also discussed extending the threat mitigation knowledge base using ontology. The CTI sharing framework, STIX was explored as the common CTI sharing approach to map various third party database together. Finally, a threat mitigation ontology that is based on MITRE ATT&CK Enterprise Network domain matrix was presented.

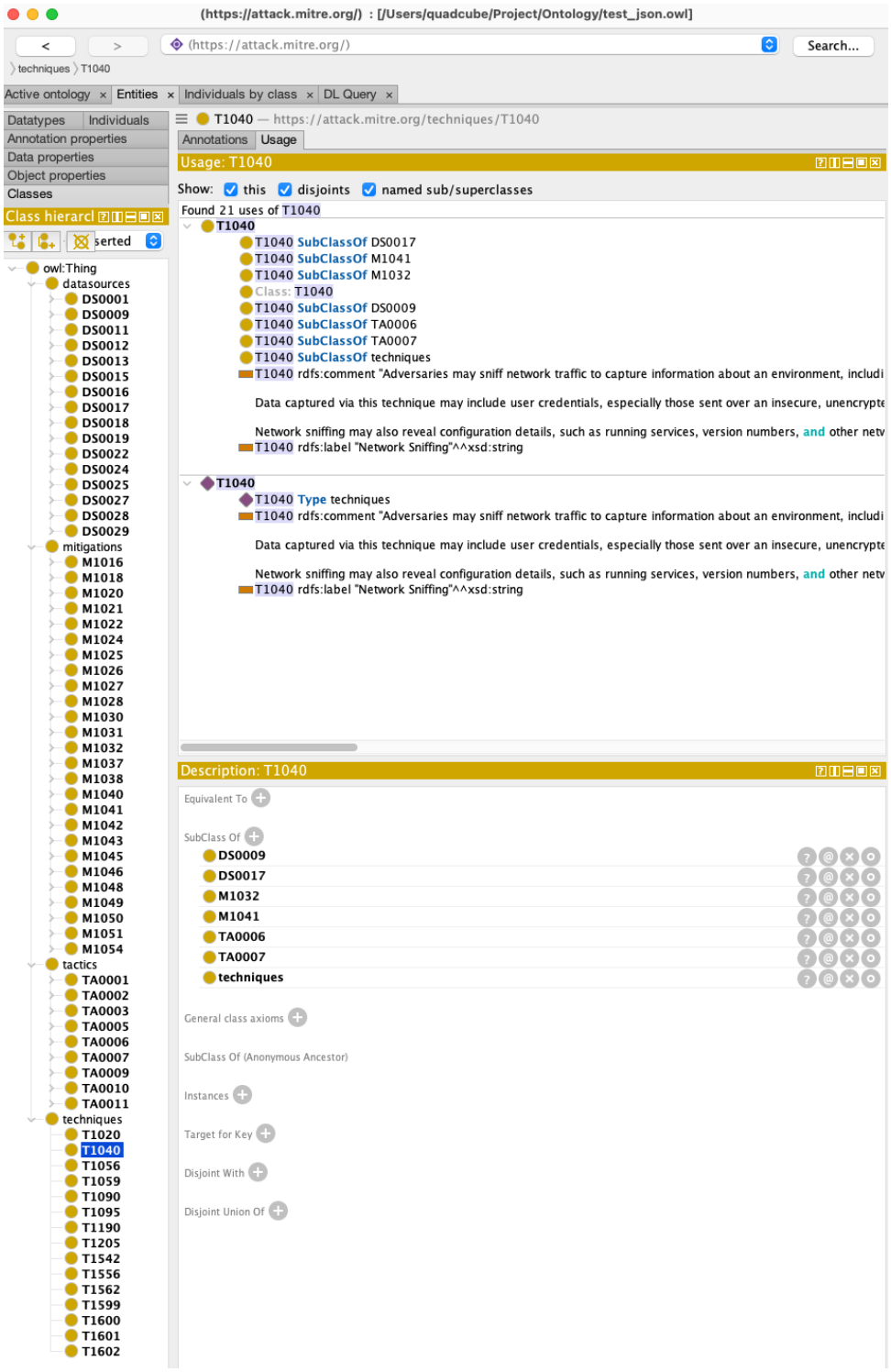

Figure 4.7: Screenshot of the MITRE ATT&CK Enterprise network domain ontology.

Algorithm 2 Generate the MITRE ATT&CK Enterprise network domain representation in JSON-LD format.

Input: STIX bundle, *bundle*; Ontology base structure, *onto_base*
Output: JSON-LD result, *jsonld*

```
1: function GENERATE_JSON-LD(bundle, onto_base)
2:     nw_techniques = get_techniques_by_platform("Network")
3:     for all indiv_technique in nw_techniques do
4:         tactic1 = ["@id": prefix + "techniques"]
5:         for all tactic in indiv_technique.kill_chain_phases do
6:             if tactic.phase_name not in tactics_all[] then
7:                 tactics_all[] ← tactic.phase_name
8:             end if
9:             tactic1 = get_tactic(tactic.phase_name)
10:            tactics1 ← ontology data entry for "tactics"
11:        end for
12:        technique_mitigations = get_mitigations_by_technique_uuid(nw_technique.id)
13:        for all technique_mitigation in technique_mitigations do
14:            if technique_mitigation.external_id not in mitigations_all
   then
15:                mitigations_all ← technique_mitigation.external_id
16:            end if
17:            tactics1 ← ontology data entry for "mitigations"
18:        end for
19:        data_srcs = get_datasources_id_by_technique_uuid(indiv_technique.id)
20:        for all data_src in data_srcs do
21:            if data_src.external_id not in datasrcs_all then
22:                datasrcs_all ← data_src.external_id
23:            end if
24:            tactics1 ← ontology data entry for "datasources"
25:        end for
26:        onto_base ← ontology data entry for indiv_technique
27:    end for
28:    for data_src in datasrcs_all do
29:        data_src = get_datasource_by_id(data_src)
30:        onto_base ← ontology data entry for data_src
31:    end for
32:    for mitigation in mitigations_all do
33:        mitigation = get_mitigation_by_id(mitigation)
34:        onto_base ← ontology data entry for mitigation
35:    end for
```

```
36:     for tactic in tactics_all do
37:         tactic = get_tactic(tactic)
38:         onto_base ← ontology data entry for tactic
39:     end for
    return onto_base
40: end function
```

Chapter 5

Automated Secure System Designer: SecureWeaver

In this section, the automated secure system designer SecureWeaver, a security extended system designer that is based on the existing Weaver, is introduced. First, the details of SecureWeaver's security verification mechanism are presented and the threat retrieval process from the service requirement is described. Next, the implementation of the seven security verification functions that correspond to the characteristics introduced in Section 4.2.1 is presented. These security verification functions provide full verification coverage of the network domain in the MITRE ATT&CK Enterprise matrix. The contents of this chapter is a part from the publications [72, 73].

5.1 Mechanism Overview

SecureWeaver is implemented as a security extension of Weaver, which take advantages of existing Weaver topology verification stage to further verify concrete topology candidate that is otherwise output by Weaver as the solution. This extension is illustrated in Fig. 5.1, where Weaver first loads the intent file and begin the refinement process to transform the abstract intent into a concrete system design. If the topology candidate is verified as concrete, SecureWeaver security verification will begin its verification process.

To address the issue of threat mitigation that was introduced in Section 4.1, a security verification algorithm which ensures all the threats specified in the intent/service requirement are mitigated prior to Weaver completing the system design is needed. At present, this thesis only considers the threats that are explicitly defined in the service requirement, and a system design is deemed secure subjected to all the defined threats are mitigated in accordance to the required security level policy.

SecureWeaver first checks that the system design candidate contains no abstract entities (components and relationships). If the system design candidate is deemed concrete, the security verification will take place in

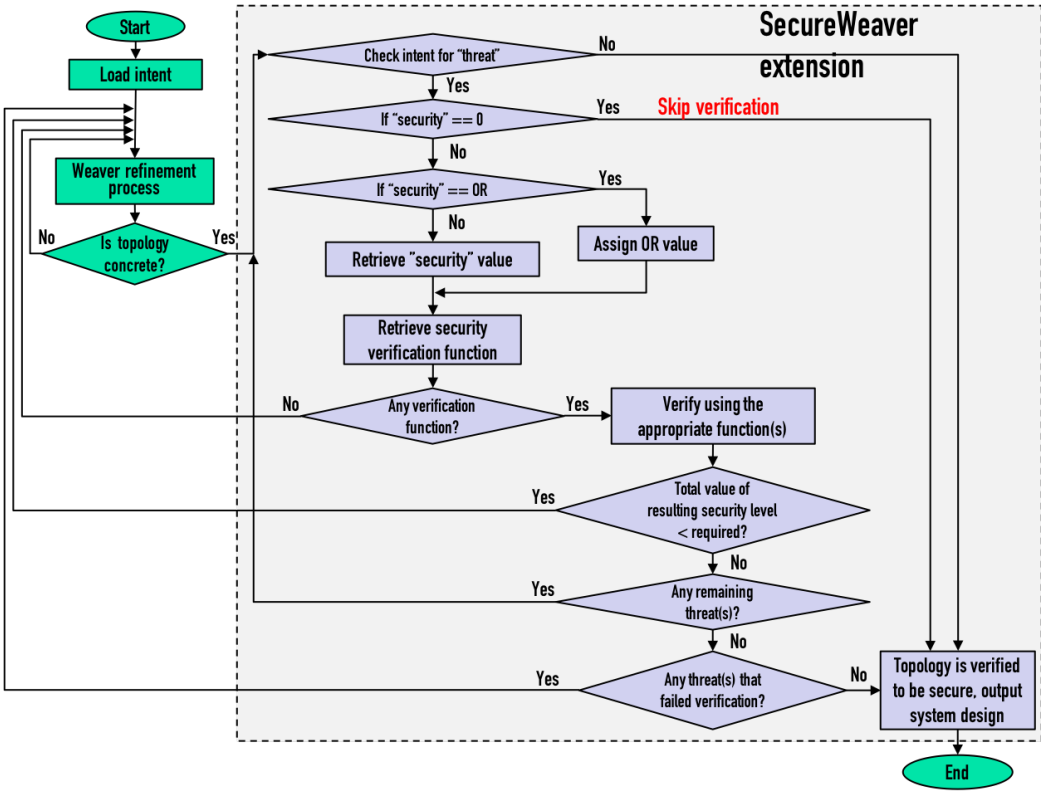

Figure 5.1: Flowchart of SecureWeaver mechanism.

two stages: (i) retrieving the explicitly defined threats from the service requirement, and (ii) dynamically calling the applicable security verification functions for system design verification. Next, the process is described in detail, followed by a description of each security verification function in Section 5.2.

5.1.1 Retrieving Threats from the Service Requirement

Since the threat(s) are defined in the service requirement and existing Weaver do not have the mechanism to "propagate" the threat in the design topology, the threat information are retrieved from the service requirement. Hence, the initial procedure of the security verification is to search in the service requirement input, t_0 for all the explicitly defined component and relationship type threats, n_{threat} and e_{threat}, and stores them into their respective threat arrays. Then each entry in the component and relationship type threat arrays is independently verified from a mitigation perspective. All threats in

the service requirement are verified for their mitigations by default, unless explicitly marked to be omitted. All threats that are to be actually verified are then assigned to the second part of the security verification process.

Besides that, each threat in the service requirement must specify how a threat is considered as mitigated or not. It is designed as such because a particular threat may have more than one mitigation. In SecureWeaver, three verification modes were offered to check whether the defined threats were negated or not:

- OR type verification
- AND type verification
- Verification based on a configurable security level

To implement this concept, each threat-mitigation pair (e.g., M1041.T1040) is assigned a weight, which denotes how much of a certain mitigation method mitigates a threat. The OR type verification mode denotes that at least one mitigation method should be applied for the threat to be considered as negated, thus it is equivalent to a minimum security level. In SecureWeaver, the minimum security level is implemented by comparing the total accumulated weight against a threshold equal to the minimum value of `float` numbers in Python 3 (sys.float_info.min). The AND type verification mode denotes that all available mitigations related to the particular threat must be met in order for that threat to be considered as neutralized. This is equivalent to a maximum security level, which is implemented in SecureWeaver by checking the total accumulated weight against a threshold equal to the maximum value 1.0.

The configurable security level verification mode denotes that a threat is considered to be negated if the total accumulated weight is larger or equal than the required/configured security level, which is any value between 0.0 and 1.0. As mentioned earlier, the security verification for a certain threat can be disabled by explicitly defining its configurable security level to the value 0.0.

An example of a simple thin client system service requirement with a threat defined is shown in Fig. 5.2, where the MITRE ATT&CK "Proxy" (`T1090`) threat is defined in the `ThinCient` type component with the maximum security level 1.0 (AND type verification) under its properties.

5.1.2 Calling the Security Verification Functions

The second part of the security verification process is where the actual system design is verified. The process first determines the target threat corresponding security verification functions, then executes them accordingly.

Test-TC-Sec.json


```
{
    "intent": {
        "nodes": [
            {
                "id": "Management_LAN",
                "type": "LAN"
            },
            {
                "id": "ThinClientSystem_1",
                "type": "ThinClientSystem"
            },
            {
                "id": "ThinClient_1",
                "type": "ThinClient",
                "properties": {
                    "threat": "T1090",
                    "security_requirement": 1.0
                }
            }
        ],
        "edges": [
            {
                "type": "belongTo",
                "src": "ThinClientSystem_1",
                "dest": "Management_LAN"
            },
            {
                "type": "belongTo",
                "src": "ThinClient_1",
                "dest": "Management_LAN"
            },
            {
                "type": "connTo",
                "src": "ThinClient_1",
                "dest": "ThinClientSystem_1"
            }
        ]
    }
}
```


Figure 5.2: Service requirement example that includes the threat T1090.

In order to determine appropriate security verification functions for the target threat, the algorithm will retrieve all the mitigations applicable to the given threat from the secure design database, and store their relevant information in an array. SecureWeaver can also be specified explicitly on the exact mitigation to be considered for verification, which in this case, SecureWeaver will retrieve the specific security verification function that is mapped to the given threat and mitigation pair.

After retrieving all the relevant security verification functions, the algorithm will first verify if the array is not empty. If the array is empty, this denotes that there is no known security verification function that cloud be determined from the security design database, and the algorithm will return the value `False`. From a security verification point of view, the return value of `False` is equivalent to rejecting the selected topology state.

If the array is not empty, the threat-mitigation pair information in the array will be individually parsed for their entity type as different sets of input arguments is required for the security verification functions depending on whether its a component or relationship type threat. After the type of threat is identified, the algorithm will dynamically call the corresponding security verification function via its name, and pass to its input all the required arguments. At this point, the security verification function will perform the actual verification and return its result.

The algorithm will retrieve the security level weight that is assigned to the target threat-mitigation pair if the security verification function returns `True`. The security level weight is then summed to the current total weight for the target threat. On the other hand, if the security verification function returns `False`, then the current total weight is not updated. The total accumulated weight is rounded off to two decimal places after verifying all the mitigations with their respective security verification function to prevent numerical float calculation errors especially in AND verification mode.

Finally, depending on the security verification mode (OR, AND or security level-based), the algorithm determines whether the total accumulated weight for the target threat is larger or equal to the threshold mentioned in Section 5.1.1. The second part of the security verification process will return `True` if this condition is met, thus the corresponding threat is considered as mitigated. On the contrary, the selected topology state will be discarded if the second part of the security verification returns `False`. In this case, the next topology state produced by Weaver iteratively verified until a secure topology state is found.

5.2 Security Verification Functions

In this section, the security verification functions introduced in Section 4.2.1 are individually described in detail.

5.2.1 Application Isolation and Sandboxing Verification

The main objective of the MITRE ATT&CK mitigation "Application Isolation and Sandboxing" (M1048) is to restrict the execution of code to a virtual sandboxed environment or in-transit to an endpoint system. This mitigation can be verified by SecureWeaver by checking for the existence of a concrete component type like `VirtualMachine` in the topology state.

The isolation/sandboxing verification function is implemented in SecureWeaver by calling a more generic subroutine, `verify_type()`, which searches for the given target node while traversing the topology from the source node affected by the target threat. Specifically, this subroutine as shown in Algorithm 3 verifies whether the component of the target type, n_{type}, is a child of the input source component, n_{src}.

Line 2 of Algorithm 3 shows the subroutine first retrieves all the relationships from n_{src} to other components, and stores them into the array $n_{conn}[]$. For each component relationship n_{conn} in $n_{conn}[]$, the subroutine determines whether the component's relationship type contain the "wire:" prefix. The relationship "wire:" is defined Weaver as an internal connection. If the condition is met, n_{conn}'s destination component is then retrieved and assigned to the destination component variable, n_{dst}. Then, on Line 6, the destination component type, $n_{dst,type}$, is checked, and if $n_{dst,type}$ is found in the n_{type} (as a list) or $n_{dst,type}$ is equivalent to n_{type} (as a single variable), the subroutine will return the value `True`. This denotes that the searched component type is in fact a child of the input source component. The syntax "in" in Python is employed in this paper such that it is utilized to check if a value exists in a sequence, such as a list or string.

If the check on Line 6 fails, the subroutine will check whether the $n_{dst,type}$ is not equal to "wire:lan". If the condition is satisfied, the type verification subroutine is then called recursively with the component relationship destination, $n_{conn,dst}$, as the next input source component. The type verification will explore all internal Weaver connections of related to the input source node except the components that are connected with relationship type "wire:lan", which denotes an external LAN connection instead of an internal connection. If the target type is not found after exhaustively exploring all components, i.e., the $n_{conn}[]$ array is empty, the subroutine will return the value `False`.

Algorithm 3 Verify existence of a node type.

Input: Topology, t; Source node, n_{src}; Node type, n_{type}
Output: True or False

```
1: function VERIFY_TYPE(t, n_src, n_type)
2:     n_conn[] ← all connected edges to n_src
3:     for all n_conn do
4:         if "wire:" in n_conn,type then
5:             n_dst ← node n_conn,dst
6:             if n_dst,type in n_type then
7:                 return True
8:             end if
9:             if n_conn,type != "wire : lan" then
10:                return verify_type(t, n_conn,dst, n_type)
11:            end if
12:        end if
13:    end for
14:    return False
15: end function
```

The subroutine `verify_type()` discussed above is called in the application isolation and sandboxing verification function, where its arguments are:

1. The component affected by a threat as the input source component
2. `VirtualMachine` as the target component type

5.2.2 Firewall Use Verification

A MITRE ATT&CK threat such as "Exploit Public-Facing Application" `T1190` can be verified via the firewall use verification function that determines whether the corresponding mitigation such as Exploit Protection (`M1050`) is applied in the topology state.

The mitigation `M1050` states that in order to safeguard a system with public-facing component such as web application (`WebApp`) against exploits, a firewall or equivalent component/mechanism is required. Hence, the implementation of the firewall use verification algorithm is as the following:

1. Search for child-of target network component
2. Network-specific mitigation verification

The first part of the process is implemented as a child-of target component specifically for network components, named `search_nw_node()`, as shown in Algorithm 4. While `search_nw_node()` subroutine is mostly similar to the

type verification introduced in Algorithm 3, the main difference between them is that Algorithm 3 only traverse within a single physical machine excluding the network section (before $wire : lan$ relationship type), whereas Algorithm 4 traverse the child components inclusive of network components until it locates a valid type of network component. In Algorithm 4 Line 8, every $wire : lan$ relationship type that is located are determined whether the destination of the $wire : lan$ relationship is a valid type of network component such as $L3SW$ and $L2SW$. If a valid network component type is found, the selected $wire : lan$ relationship is then stored in a list, $connected\_nw[]$ as shown in Algorithm 4 line 10 to 11.

Next, the `search_nw_node()` subroutine will search exhaustively for the rest of the $wire : lan$ relationships and add them into the list. The algorithm will recurse with destination component of the relationship $n_{conn,dst}$ to find the following child component if the component is not the valid type as shown in Line 13.

After processing all the relationships in $n_{conn}[]$, the algorithm will check whether the array $connected\_nw[]$ is not empty. If the condition is met, the algorithm will return $connected\_nw[]$ array indicating a successful search process. On the other hand, if the array is empty, the algorithm will return `False` denoting the child-of target component does not have any relationship to any network components.

If the first part of the firewall use verification (search process) is successful, the result is passed to another verification subroutine `verify_nw_node_type()`. This subroutine is shown in shown in Algorithm 5 where it verifies the mitigation based on the type of the network component. First, the subroutine iterates though every network relationship, $connected\_nw$ in the $connected\_nw[]$ array returned by Algorithm 4. The relationship(s) of each network relationship destination, $connected\_nw_{dst}$ are retrieved and stored in e_{NW} as shown in Line 3. All relationships that are stored in e_{NW} are then processed, where the source of the object's relationship is retrieved and store into n_{NW} as shown from Line 4 to 5. On Line 6, the n_{NW} type is then determined whether the type is found in the n_{type} input, such that n_{type} can be either single type or a list of types which increases the flexibility of the algorithm input handling. The algorithm will return `True` if the condition is met indicating that the target type is a part of the affected component network. On the other hand, the algorithm will retrieve all relationships related to the e_{NW} relationship source object if the condition is not satisfied, and store it into an array, $connected\_NW1[]$ as shown in Line 9. Following, the $connected\_NW1[]$ provided as the new input array into the algorithm to be recurse for further verification.

Therefore, a wrapper function is implemented to verify firewall type com-

Algorithm 4 Search and verify the child-of a target component for network related verification.

Input: Topology, t; Source node, n_{src}
Output: True or False

```
1: function SEARCH_NW_NODE(t, n_src)
2:     n_conn[] ← all connected edges to n_src
3:     for all n_conn do
4:         if "wire:" in n_conn,type then
5:             if n_conn,type != "wire : lan" then
6:                 return search_nw_node(t, n_conn,dst)
7:             end if
8:         else
9:             n_NW ← node n_conn,dst
10:            if n_NW,type is a valid switch type then
11:                connected_nw[] ← n_conn
12:            else
13:                return search_nw_node(t, n_conn,dst)
14:            end if
15:        end if
16:    end for
17:    if connected_nw[] != ∅ then
18:        return connected_nw[]
19:    else
20:        return False
21:    end if
22: end function
```

ponent, which first calls the Algorithm 4 subroutine to obtain the networks the target component is connected to, followed by Algorithm 5 subroutine with the result of Algorithm 4 as its argument to determine whether a `Firewall` type component is apart of the target component network.

5.2.3 Network Segmentation Verification

Network segmentation is a technique used to separate vulnerable services and resources from the rest of the system components. For example, the "Network Segmentation" (`M1030`) mitigation from the ATT&CK framework which addresses threat `T1190` recommends segmenting external facing services and servers via methods such as Demilitarized Zone (DMZ) to isolate them from the rest of the network.

Algorithm 5 Search and verify the type of mitigation based on network component.

Input: Topology, t; Connected network, $connected\_nw[]$; Node type, n_{type}
Output: True or False

```
1: function VERIFY_NW_NODE_TYPE(t, connected_nw[], n_type)
2:     for all connected_nw[] do
3:         e_NW ← all connected edges to connected_nw_dst
4:         for all e_NW do
5:             n_NW ← node e_NW,src
6:             if n_NW,typ in n_type then
7:                 return True
8:             else
9:                 connected_NW1[] ← all connected edges to e_NW,src
10:                if verify_nw_node_type(t, connected_NW1[], n_type) then
11:                    return True
12:                end if
13:            end if
14:        end for
15:    end for
16:    return False
17: end function
```

The conditions that are used to verify whether a component is segmented or not from a network point of view are shown in Fig. 5.3. Example 1 shows the case in which a component with a threat associated to it is connected to two LAN switches, and some other components are connected to one of the LAN switches. In such a case, the component under threat is defined as a no network segmentation scenario. Note that a component under threat connected to only one network is also deemed as having no network segmentation if the total number of networks in the entire topology is one, as shown in Example 2.

Example 3 in Fig. 5.3 illustrates a segmented network topology state which has two LAN components with other types of components connected to them, but no component is connected to both LANs. Example 4 illustrates one component that is connected to two LANs, whereas the component under threat is connected to only one LAN. This case is also considered to be network segmented, since it is not possible to directly access another LAN from the component under threat without compromising the component that is connected to both LANs.

To implement the network segmentation verification algorithm a wrapper

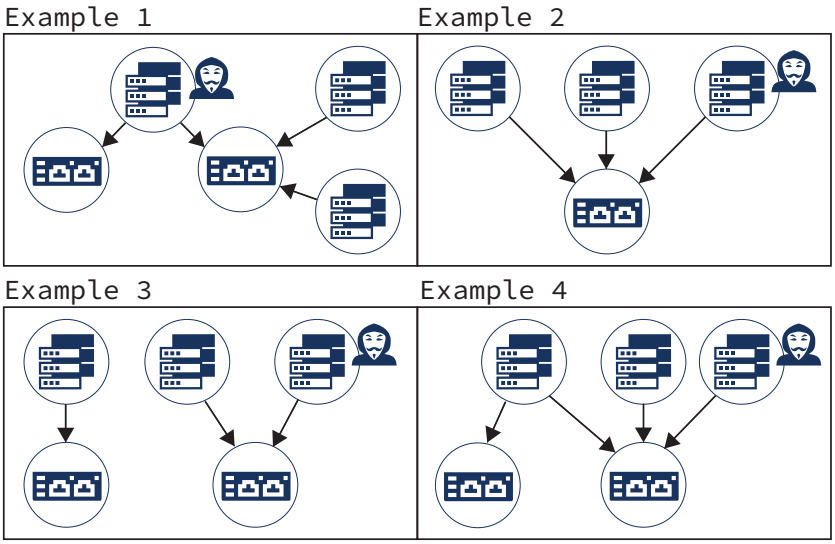

Figure 5.3: Examples of possible topology states for network segmentation verification.

function was implemented, which calls first the Algorithm 4 subroutine to determine which networks the input source component is connected to. The array of connected networks returned by this subroutine returns is then processed by a second subroutine that checks the existence of network segmentation for the input source component.

This second subroutine will first check the network type component relationships that are stored in *connected_nw*[], which are the network connections from the initial input source component identified by the first subroutine. If there is more than one network type component relationship in *connected_nw*[], it means that the input source component is connected to more than one network segments, and the subroutine will return `False`. Otherwise the subroutine will return `True`, except for the case when there is only one network in total, when it returns `False`, as discussed for Example 2 in Fig. 5.3.

5.2.4 Configuration Settings Verification

Some of the threats in the ATT&CK framework are mitigated via specific software or hardware-related configuration settings and actions, such as "Privileged Account Management" (`M1026`) or "Code Signing" (`M1045`). Since the current SecureWeaver focuses on networked system architecture design, such configuration-related mitigations are implemented as user-definable assumptions in the secure design database, as it was explained

in Section 4.2.2. Users may define whether a specific software or hardware-related configuration can be assumed to be present or not in the actual implementation of the designed system. Note that the system administrators should ensure the assumed configuration mitigation is actually implemented when the designed system is deployed.

The configuration verification function retrieves the configuration mitigation assumptions from the secure design database based on the provided threat-mitigation pair and checks whether the setting is defined as `True`. If this is the case, the value is appended to a dictionary of configuration mitigation assumptions for the component under threat (creating this dictionary if it doesn't exist yet). When SecureWeaver verifies that a given topology state is secure, it will take into account the configuration mitigation status for that state.

5.2.5 Traffic Filtering Verification

Some network-related threats, such as Man-in-the-Middle (MITM), Denial of Service (DoS), unsanctioned proxy, and many more can be mitigated via traffic filtering. This typically requires the use of a software firewall or/and specialized network appliances to filter the ingress/egress traffic. The type or combination of network traffic filters that is required to mitigate a threat is determined by the mitigation description for that particular threat, interpreted by an expert who will input the corresponding notation in the secure design database.

To verify for hardware-based network-filtering mitigation, the verification function must check that there is a valid type of network appliance for network filtering such as a `Firewall` component. On the other hand, to verify for a software firewall and its configuration settings, the verification function must check the assumed configuration settings in the secure design database. Hence, this requires the verification function to choose or combine the network type verification introduced in Section 5.2.2, and the configuration settings verification discussed in Section 5.2.4 to determine whether a mitigation is suitable for a particular threat.

The corresponding verification function is implemented as a wrapper function that calls both the aforementioned verification functions. The wrapper function first checks the secure design database for the type of the verification method: (i) hardware; (ii) software; (iii) require either one (OR); and (iv) require both (AND). Then the wrapper function calls the related verification function according to the required method to mitigate the selected threat. The hardware-based component type to be verified in the selected topology state by the network type verification is the `Firewall`

component type, while the configuration settings function will verify whether the software firewall configuration assumption is set to `True` in the secure design database. If the function calls are successful according to the required mitigation method, then the traffic filtering verification function will return `True`.

5.2.6 Secure Protocol Use Verification

While most threats in the ATT&CK framework apply to components, there are also instances of threats that refer to relationships between components. For example, the network communication between two endpoints may be susceptible to sniffing, as an adversary may be able to capture valuable information if the connection is not secured sufficiently. In order to model such a scenario, the logical and conceptual connections is introduced in Section 4.1.2, and the corresponding refinement rules in Section 4.1.3 to ensure that the designed system is secure even for in-transit data.

To verify whether a conceptual connection represented by a group of logical connections combination is secure, a secure protocol verification function is implemented where it checks the security of the application layer and network layer protocols in a given topology state. The verification function, shown in Algorithm 6, takes the component source and destination of the relationship (edge), e_{src} and e_{dst}, and the relationship threat, e_{threat}, information as the input. The algorithm determines all the relationships connected to the relationship source component, e_{src}, and stores them in the array $e_{conn}[]$, then each array element, e_{conn}, is processed. If the type of e_{conn} has the prefix "wire:" but is not "wire:lan", it means that particular e_{conn} is not a logical connection; then the algorithm will call itself recursively with the destination of e_{conn} replacing the initial e_{src}, so as to verify the child of the source component in e_{src} as shown on Lines 4–5. If the result of the recursion is `True`, the entire algorithm will return `True`, denoting that a valid mitigation has been verified.

If the condition on Line 4 is not met, this means that the selected relationship is a logical connection. Then the algorithm checks whether the destination of the selected relationship is equal to the input e_{dst}, as shown on line 9. If this is the case, it means that the logical connection is an application layer protocol in the TCP/IP model. If the condition on line 9 is not met, it means that the logical connection is a network layer protocol. The actual protocol for the logical connection is then verified with reference to the secure design knowledge base, as shown on lines 10 and 15. If the protocol for the logical connection is found in the secure design knowledge base, the secure protocol verification function will return `True`, since only

Algorithm 6 Search and verify the mitigation of threats in conceptual connections

Input: Edge source, e_{src}; Edge destination, e_{dst}; Edge threat, e_{threat}
Output: True or False

```
1: function VERIFY_CC(e_src, e_dst, e_threat)
2:     e_conn[] ← all connected edges to e_src
3:     for all e_conn do
4:         if e_conn,type prefix == "wire:" and e_conn,type ! = "wire:lan" then
5:             if verify_CC(e_conn,dst, e_dst, e_threat) then
6:                 return True
7:             end if
8:         else
9:             if e_conn,dst == e_dst then
10:                CC_mitigation ← e_conn mitigation info
11:                if CC_mitigation,threat == e_threat and CC_mitigation ! = ∅ then
12:                    return True
13:                end if
14:            else
15:                CC_mitigation ← e_conn mitigation info
16:                if CC_mitigation,threat == e_threat and CC_mitigation ! = ∅ then
17:                    return True
18:                end if
19:            end if
20:        end if
21:    end for
22:    return False
23: end function
```

secure protocols are recorded in the knowledge base.

5.2.7 Intrusion Detection and Prevention System (IDPS) Use Verification

While firewalls may limit the inbound or outbound access between networks, firewall filtering rules must configured in advance for this purpose. For inbound traffic, in particular, a "hole" must be opened to allow the traffic pass through the firewall. A Network Intrusion Detection System (NIDS) or an Intrusion Detection and Prevention System (IDPS) is often deployed in conjunction with a firewall, and its functions is to analyze the traffic based on an internal database that contains intrusion detection signatures of

attacks on specific applications. When a suspicious activity is detected, an NIDS will typically send an alert to the management terminal for the system administrator to take further action, whereas an IDPS will log the attempt and actively try to prevent the attack.

In order to verify whether an NIDS or IDPS are present in a given topology state, a wrapper function is implemented to search for `NIDS` or `IDPS` type components. This function calls the two subroutines shown in Algorithm 4 and Algorithm 5, providing the list of target components as the target type input. If either an `NIDS` or `IDPS` type component is found on the path traversing between two networks, the IDPS use verification function will return `True`.

5.3 Summary

In this chapter, the verification mechanism of SecureWeaver and its security verification functions were presented. Details of the security verification mechanism were presented first, thus, the threat retrieval process from the input service requirement was described, followed by the description of dynamic calling process of security verification function. Besides that, the seven security verification functions that provide full verification coverage of the Network domain in MITRE ATT&CK Enterprise matrix were described in detail.

Chapter 6

Secure System Implementation Case Studies

In this chapter, several case studies on secure system implementation are explored. The secure system implementation is explored using an IoT hardware platform, MkIoT, where the first case study investigates the implementation of secure end-to-end communication, and the second case study explores the implementation of secure configurations. The results from the case studies are used as motivating evaluations for the thesis. The content of this chapter is a part from the publication [93] (note that the paper includes an additional case study about IoT life-cycle management).

6.1 Hardware Platform Design

A set of design requirements is typically utilized in the design phase of a hardware platform. The typical requisites that should be considered in the design decision [94, 95], are as follows:

- Ease of development
- Data acquisition, processing and storage
- Connectivity
- Power
- Cost
- Security

Ease of development looks into factors such as the availability, accessibility, and quality of documentations and tools relevant to the development of a hardware. Data acquisition, processing and storage are generally the quantitative requirements that is defined by the target application for the hardware, as well as power requirements and cost. The target application also determines the type of connectivity (wireless/wired) and also the power requirement (battery/wall-powered). Last but not least, the security requirement determines the required hardware features such as hardware-

based cryptography engine, secure boot, and flash encryption. The design requirements will be utilized to describe the features of hardware platform that used in the case studies.

6.2 Hardware Platform Implementation

With the purpose of being representative for generic IoT applications, MkIoT should target common maker use cases and applications. The requirements presented in Section 6.1 are employed for various use cases, such as: simple smart home device, remote sensing in agriculture, asset tracking and many more applications that shares similar architecture. Hence, the microcontroller unit (MCU) used in the MkIoT should meet the requirements listed below:

- Widely used in maker communities
- Supported by Arduino
- Adequate processing capability and storage while being affordable

For connectivity in MkIoT, a common bus interface with off-the-shelf sensor and industrial modules should be present for wired connectivity, while it should also have WiFi and Low-power WAN (LPWAN) for wireless communication. Furthermore, MkIoT should be able to be powered either through wired or battery power.

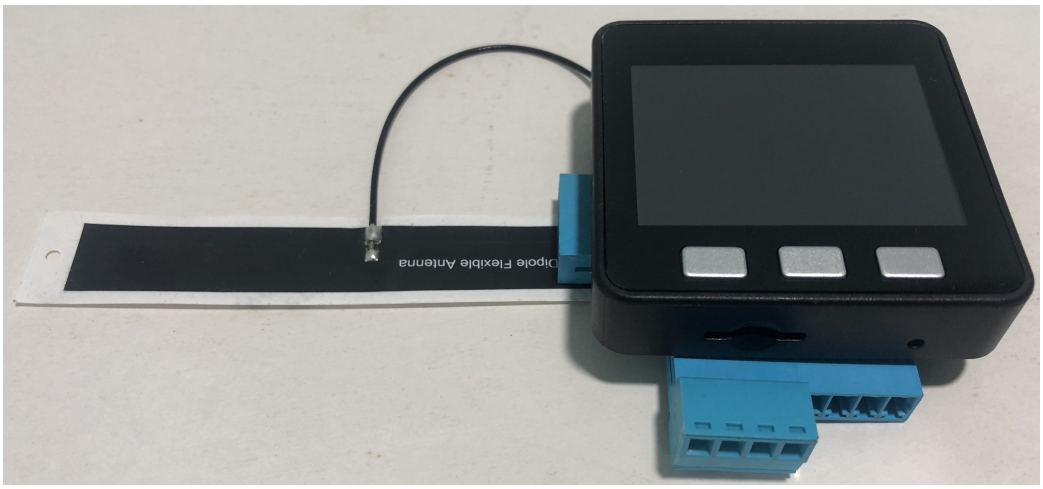

Figure 6.1: External view of the MkIoT hardware platform.

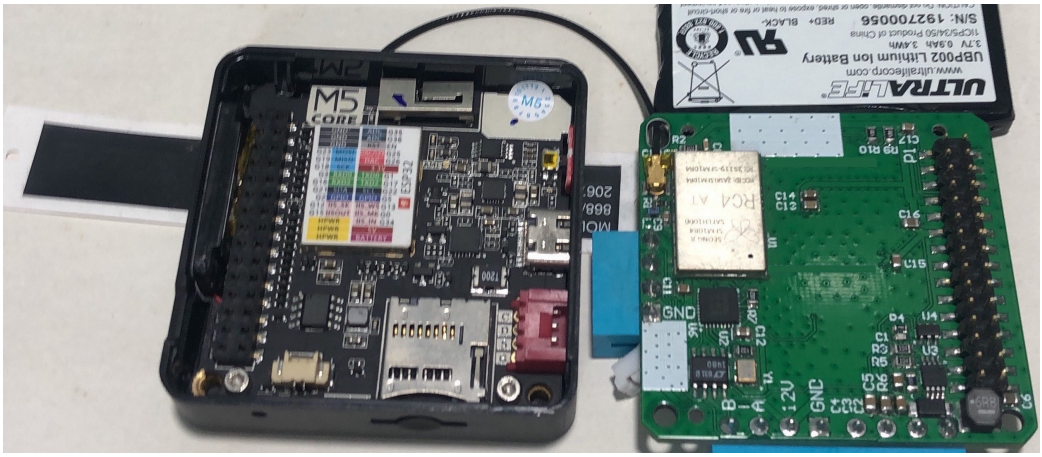

Figure 6.2: Internal view of the MkIoT hardware platform.

MkIoT is composed of two main parts, as shown in Figs. 6.1 and 6.2: (i) MCU module and (ii) expansion daughterboard. The MCU is based on Espressif ESP32, a popular MCU in the maker communities and have a large varieties of tutorial and examples which satisfies the ease of development criteria. The MCU module is an off-the-shelf module, the M5Sstack ESP32 basic Core IoT development kit, which is a relatively affordable and accessible MCU module. It features a dual-core MCU, which runs up to 600 MIPS, 448KB ROM and 520KB SRAM. Besides, it also able to store large amount of data locally as it supports multiple external flash chips through its QSPI bus. The ESP32 module is equipped with both WiFi and Bluetooth Low Energy (BLE), which partly satisfy the connectivity requirement. The M5Stack module also supports both wired and battery operation, which satisfy the flexible power requirement.

Typically, the more wired or wireless interfaces a device has, the more diverse the application domains the reference hardware platform can explore. Thus, for the design of MkIoT, emphasis is placed on both wired and wireless interfaces/connectivity.

6.2.1 Wired Connectivity

The ESP32 MCU comes standard with I2C, I2S, SPI, UART and CAN bus. To allow MkIoT to communicate via industrial standard communication, RS485, a daughterboard was developed to provide RS485 communication though both NXP SC16IS752IBS I2C-to-dual UART interface and LTC1480 for ultra-low power RS485 transceiver. Besides that, a DC step-up boost

regulator was also added to the daughterboard to provide a 12V power rail to any industrial device powering off MkIoT.

6.2.2 Wireless Connectivity

In order to support LPWAN, a Sigfox transceiver, Wisol SFM10, was included in the daughterboard design. Sigfox is selected as the candidate for LPWAN communication implementation due to the reasons as below:

1. Relatively simple device onboarding process to connect an IoT device to Sigfox network
2. High ease of development
3. Low cost
4. Low power consumption for battery use cases

An illustration of Sigfox use case is shown in Fig. 6.3. There are two SFM10 transceivers that were tested in this chapter, which are RC3 for Japan and RC4 for Asia Pacific countries, such as Malaysia. The development and testing was done first in Ishikawa, Japan and later in Kuala Lumpur, Malaysia, and in both cases E2E communication from MkIoT to the Sigfox cloud was successful. One of the drawbacks observed was the absence of signal indoors, rendering it useless in such an environment. The u-blox SAM-M8Q GPS module was also included to support asset tracking use cases.

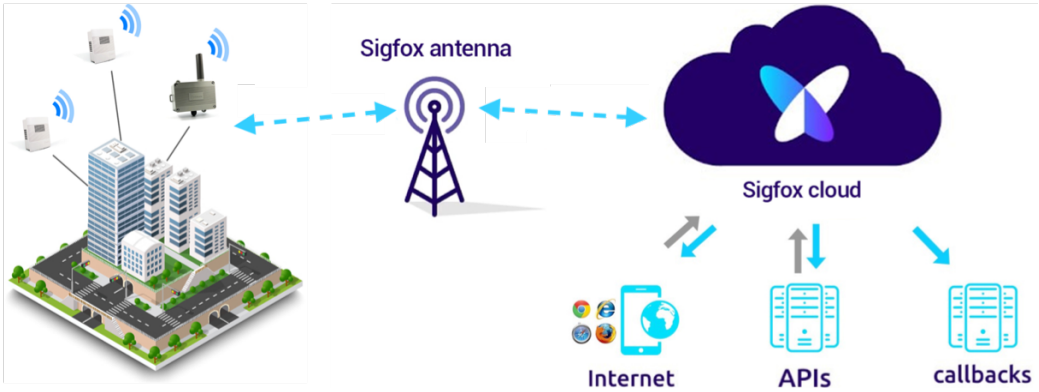

Figure 6.3: Example use case of Sigfox end-to-end communication.

The schematic for the MkIoT daughterboard is shown in Fig. 6.4, where the circuit design for the RS485, Sigfox transceiver, GPS module, and power regulation are shown.

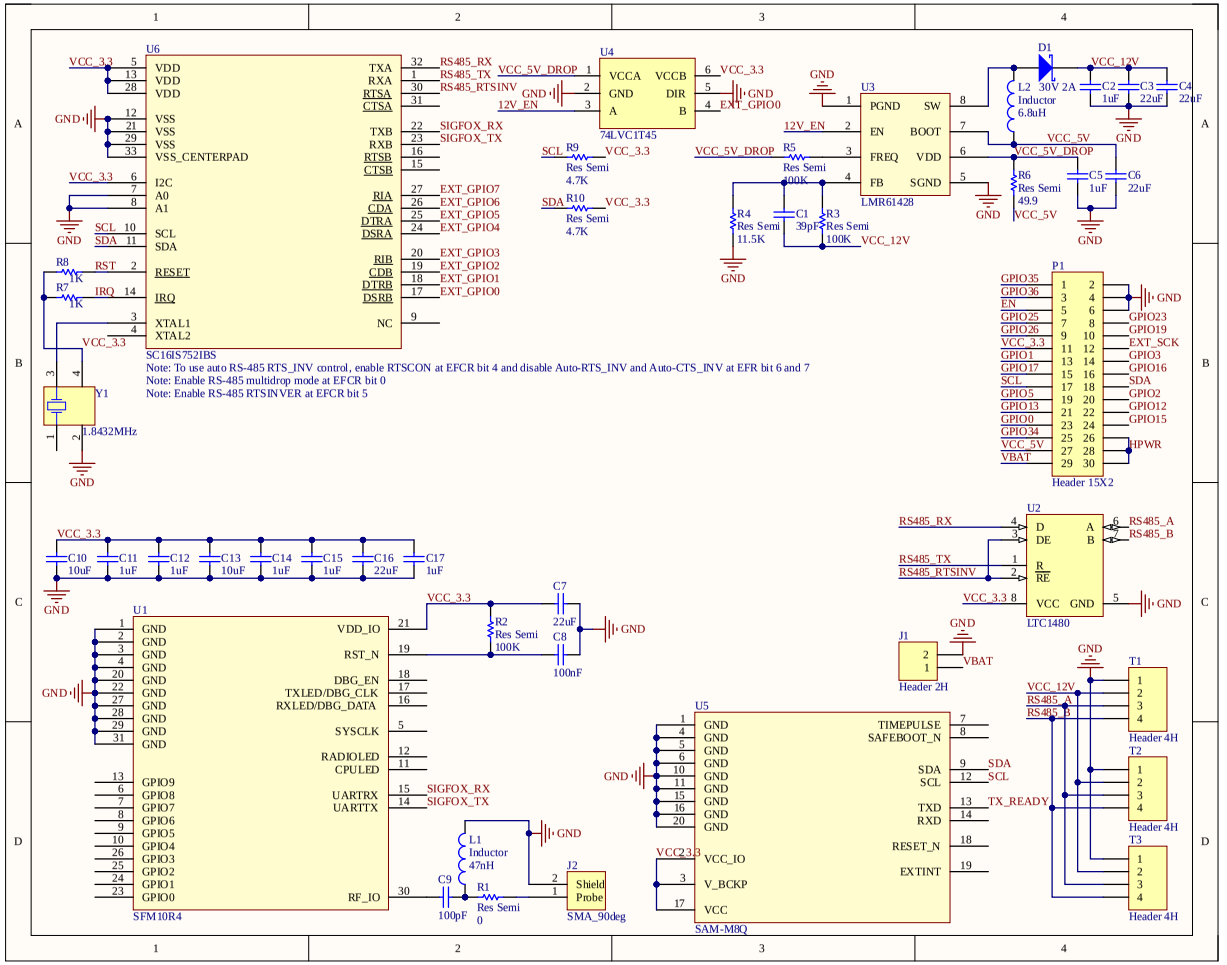

Figure 6.4: MkIoT daughterboard schematic.

6.3 Case Study #1: Secure End-to-End Communication

For case study #1, an end-to-end (E2E) application prototype is implemented, in which the IoT device sends data to or receives data from the cloud. Hence, focus is placed on the device and E2E communication integrity and security. The architectures and security practices that are relatively straightforward for a maker to implement are explored, and the pitfalls encountered throughout the process are analyzed. In this analysis, it is assumed that the public cloud provider implements best security practices to safeguard their service and infrastructure [96].

In order to implement secure E2E communication, transaction-level security is ensured by measuring integrity of the entire system and transaction, E2E. Secure E2E communication is implemented over LPWAN and IP network to the public cloud endpoint in this case study.

6.3.1 Sigfox Security

Sigfox is a lightweight protocol, suitable for very-low power remote sensor IoT application. Some use cases for Sigfox are remote sensing and asset tracking. The lightweightness of Sigfox also constrains the application to 12-bytes payload for uplink and 8-bytes payload for downlink. The total size of a Sigfox frame is 26-bytes in total, where the frame structure is illustrated in Fig. 6.5. Moreover, there is also a limit of up to 140 uplink and 4 downlink messages per day, which is still sensible for its target use case market.

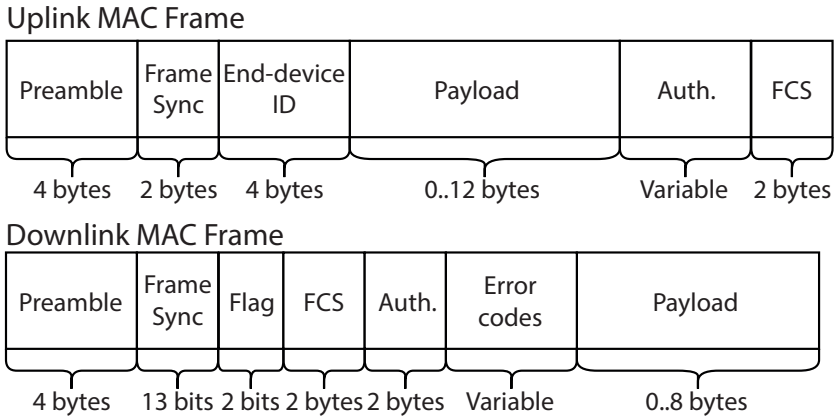

Figure 6.5: Sigfox frame structure.

For case study #1 implementation using LPWAN, two architectures were considered, with their endpoints being:

1. Sigfox cloud
2. Google Cloud Platform (GCP) through Sigfox backend

For the first architecture, the implementation was fairly straightforward, as the SFM10 Sigfox transceiver can receive AT commands over UART as long as the transceiver identity is registered at the Sigfox cloud and is authorized to communicate. Further infomation on Sigfox UID and encryption key can be found in [97]. The second architecture, as illustrated in Fig. 6.6, is actually an extension of the first, where the uplink payload is received at Sigfox cloud and a set of callbacks using Sigfox backend API is setup to forward the payload to GCP HTTPS endpoint. The transaction between Sigfox and GCP is authenticated by basic HTTP username and password over HTTPS, for which the credentials are stored in plain text on Sigfox backend. Hence, at this point, the services provided by public cloud providers are assumed to be secure.

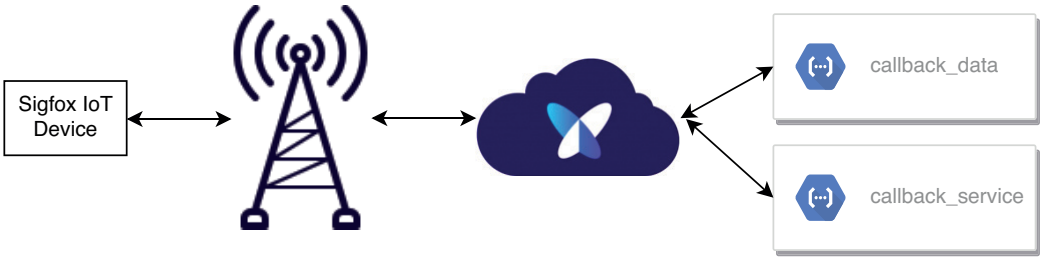

Figure 6.6: Sigfox end-to-end communication to Google Cloud Platform.

In a Sigfox network, a unique ID, the Network Authentication Key (NAK), and the encryption key are burned into the ON Semiconductor AX-SFUS-1-01 MCU inside the SFM10 transceiver module. This process relies on the trusted supply chain between the module manufacturer and Sigfox. The last two being secret, and 128 bits in length. Each message sent by a device or the cloud contains a unique signature generated using the encryption key. This signature authenticates the sender, and to make sure there are no copies or duplications possible, the system inserts it in a sequence of numbers. The transmission is unsynchronized between the devices and the network. They broadcast each message 3 times on 3 different frequencies (frequency hopping). The base stations monitor the spectrum and look for UNB signals to demodulate.

Sigfox frames are sent in plain text by default, although there is encryption support since 2017, which is provided by the Sigfox end [98]. However, there are no details about enabling payload encryption in the SFM10 datasheet. This restricts the makers to either utilize another transceiver or rollout their own payload encryption, which is an issue by itself in [97]. This issue is also covered in [99], where Sigfox is used only for the network security itself, and delegates the payload security issue to the makers.

6.3.2 MQTT Security

For case study #1 implementation using IP, the IoT device communicates over the IP network using the MQTT protocol with GCP. Before going into the implementation details, the key points on MQTT security are first discussed. Firstly, the original MQTT standard does not require authentication in MQTT as mandatory [100]. While it is possible to host an authentication-less MQTT broker in a secured network, this is not widely practised in most of the implementation tutorials and examples. Secondly, basic authentication in MQTT is not as secure as mostly thought. While authentication is supported via username and password fields in the CONNECT message, these are sent in plain text in TCP over the network, which is an eavesdropping risk. Many popular MQTT setup tutorials, especially for smart homes, are using this method, which is a risk if the maker expose the service to public network.

In order to solve this issue, MQTT can be used over TLS to encrypt the whole MQTT communication. While introducing TLS on the MQTT broker has insignificant performance penalty, that cannot be said very constrained IoT devices, where the increase in overhead processing will reduce its battery operational time. The maker has to balance the requirements in order to accommodate the overhead or settle for a less secure implementation. For public cloud provider such as GCP,TLS connection to their MQTT broker is mandatory.

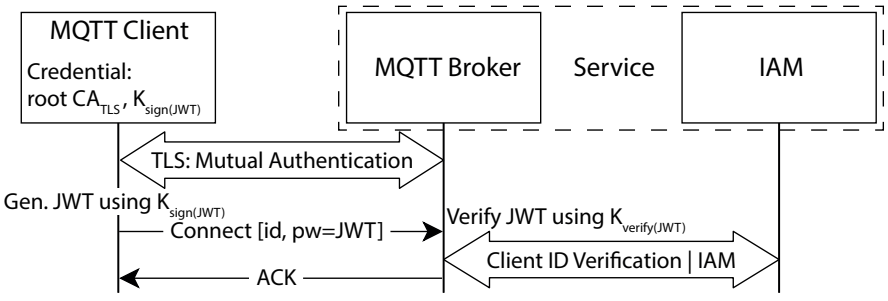

Figure 6.7: MQTT communication using TLS and JWT.

In addition to TLS, other security mechanisms can be used to increase security, for instance, JSON Web Token (JWT), which is also used in addition to TLS on GCP MQTT broker, as shown in Fig. 6.7. JWT enables per-device authentication, which limits the attack surface as compromised key would only affect a single device rather than the whole group. Besides, each JWT is only valid to up to the user setting or maximum 24 hours, which ensures that the compromised key will expire. Google maintains a sample Arduino library to make implementation simple for makers. One issue that affected many makers using the Arduino library to connect to GCP is that there is no built-in mechanism to refresh the JWT. This requires the maker to periodically disconnect and reconnect back to GCP with a new JWT, causing session interruption, which may not be ideal for certain IoT applications as unwanted latency is introduced. Another issue that was faced during case study #1 implementation was the connection rejection if the JWT timestamp does not tally with GCP side. JWT is also time dependent; hence, there is a risk of DoS as the IoT device is now dependent on NTP servers to resolve its system time.

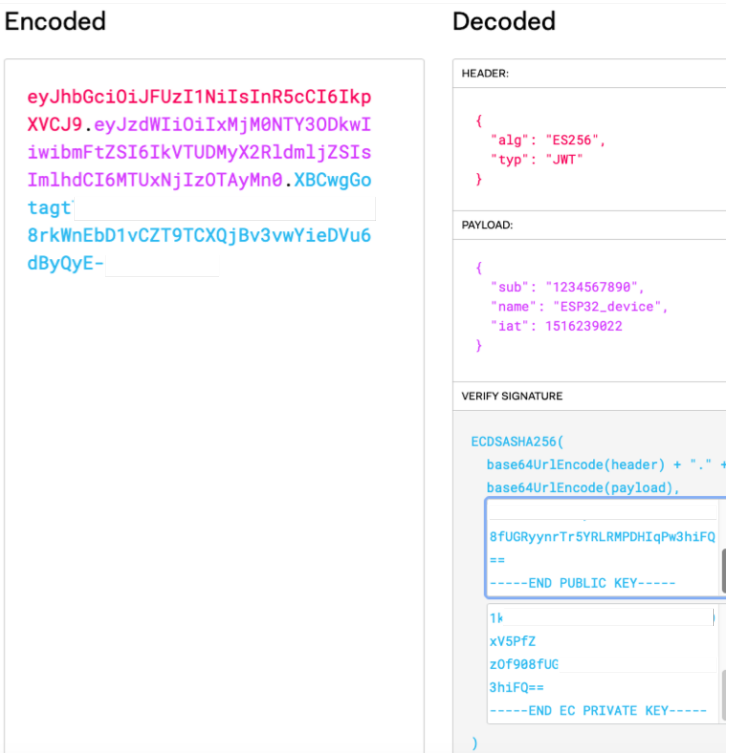

Figure 6.8: MQTT JWT token.

From this case study, it is clear that implementing a theoretically secure

communication method does not guarantee that it can mitigate against all attacks. This is due to issues related to implementation/configuration; hence, in the next case study the secure configuration is explored in view of this issue.

6.4 Case Study #2: Secure Configuration

The ESP32 MCU has some built-in security features, such as secure boot, flash encryption, 1024-bits OTP, cryptographic hardware acceleration for AES, hash (SHA-2), RSA, ECC and random number generator (RNG). This section explores and discusses on the approach of utilizing the MCU hardware security features to secure the device integrity.

6.4.1 Porting the Arduino Core into ESP-IDF

The Arduino software platform plays a very important role in the maker culture. It enables laymen to tinker with programming and electronics with little effort due to its extensive abstraction of the hardware layer into simple functions to control the MCU's I/O and perform computations. Arduino platform is also famous for its extensive software libraries, which are contributed by the open source communities. Arduino also provides an IDE to develop applications, direct firmware build and upload to the ESP32 device from the user friendly IDE. While these may be positive from an ease of development standpoint, it may also quickly turn into a security nightmare. There is a number of vulnerabilities that make Arduino easily exploitable as presented in [42]. Reference [43] also includes a case study on the Arduino vulnerability impact on IoT devices.

6.4.2 Locking Down Arduino on ESP32

The ESP32 MCU is often paired with an external flash chip that stores the user applications and data. Hence, any maker with physical access to the ESP32 based device could read the flash chip content via serial or desoldering and reading the physical flash chip directly through SPI. Firmware reverse engineering and modification could be done if the device maker does not secure the device sufficiently. Fortunately, ESP32 features hardware-based flash encryption and secure boot to prevent unwanted flash accesses. There is a caveat, however, as it requires the Espressif integrated development framework (ESP-IDF), which is mainly low-level embedded C, to enable the security features. Since the prototype implementations of case study #1

were developed using Arduino, the implementation is restricted to the usage of Arduino code, as it is difficult for a non-technical maker to jump from Arduino into embedded C. One workaround is to use the Arduino core as a "component" in ESP-IDF as shown in [101].

The Arduino code was ported using the workarounds in [101] to ESP-IDF, but many issues had to be ironed out, such as external SPI RAM failing to initialise, broken links to BLE, missing TLD setting and many more. Furthermore, the Arduino core had to be compiled to specific version of ESP-IDF, which were outdated (v3.x) compared to the latest stable (v4.x), missing out certain security features and bug fixes. Enabling the security features in ESP-IDF is trivial as shown in Fig. 6.9.

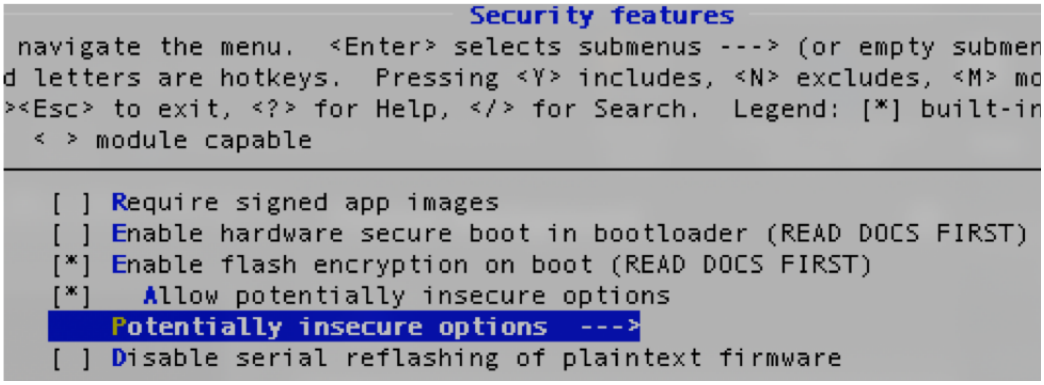

Figure 6.9: ESP-IDF security features.

The ESP32 bootloader, ported Arduino application code and custom partition table from Arduino minimal SPIFFS partition scheme were prepared. Then, an encryption key was generated and burned into the ESP32 device eFuse as shown in Fig. 6.10. Table 6.1 shows the descriptions and size of each parameter security fuse.

The unencrypted bootloader, partition table and application are then written to the ESP32, which will auto reset and encrypt the bootloader, application partitions and any partition that is flagged as encrypted (see Fig. 6.11).

While Fig. 6.11 shows some security warnings regarding insecure configurations, the ESP32 device was not fully locked down in the settings, as it would limit the ability to further analyse the flash after encryption. Moreover, if there are any errors during the flash encryption procedure, the ESP32 may be bricked, rendering the device useless.

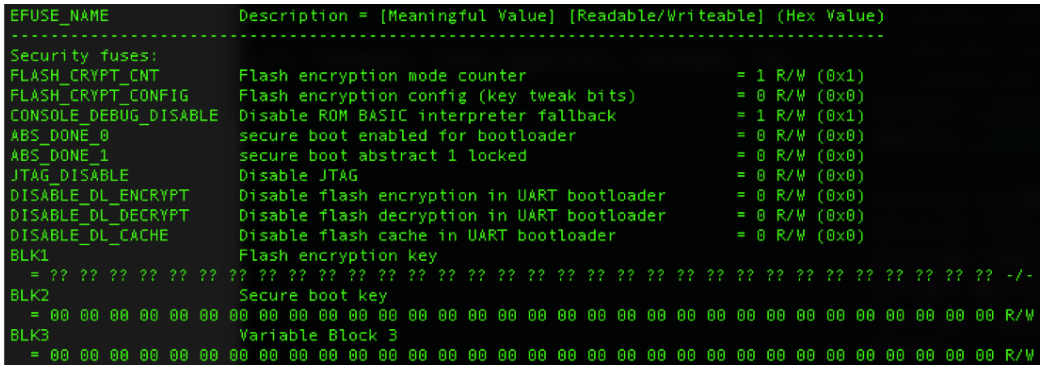

Figure 6.10: eFuse properties after enabling flash encryption and burning the encryption key into the eFuse BLK1 register.

Table 6.1: ESP32 security eFuse fields.

Name	Description	Size (bit)
FLASH_CRYPT_CNT	Flash encryption count	7
FLASH_CRYPT_CONFIG	Flash encryption config	4
CONSOLE_DEBUG_DISABLE	Disable ROM console	1
ABS_DONE_*	Enable secure boot	2
JTAG_DISABLE	Disable JTAG	1
DISABLE_DL_*	Disable in UART	3
BLK1	Flash encryption key	256
BLK2	Secure boot key	256
BLK3	Variable block 3	256

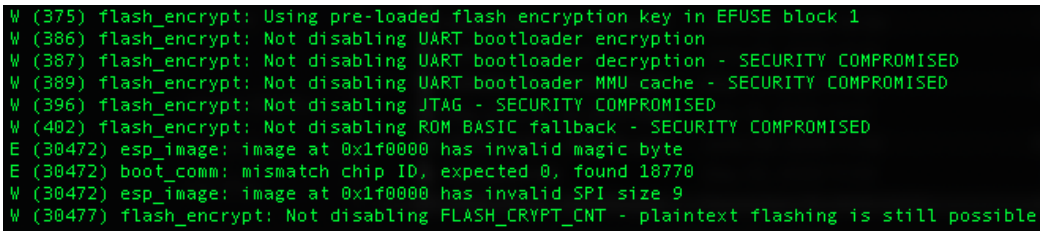

Figure 6.11: ESP32 device security issues log on first boot.

Figure 6.12: Extracted binary (bootloader section) before and after flash encryption.

6.4.2.1 Flash Chip Content

Next, the ESP32 flash chip is analyzed for both unencrypted and encrypted scenarios. For ESP32 series, all except ESP32 pico have the firmware stored on an external memory chip. The flash chip content is read through ESP32 serial for both scenarios, where a sample of the bootloader before and after encryption is shown in Fig. 6.12.

However, through analysis it was discovered that not every partition is encrypted. By default, only the bootloader, partition table, OTA data and App related partitions are encrypted [101]. The prototype code that performs secure E2E communication utilized the Arduino EEPROM library to store its private keys in the flash. The earlier version of the EEPROM library reads and writes contents into a fixed partition block where the recent version emulates the read write functions and store the data in the non-volatile storage (NVS) partition. NVS partition is not directly compatible with flash encryption while fixed EEPROM partition block is unencrypted by default. Hence, the application private keys can still be read as plain text, as shown in Fig. 6.13.

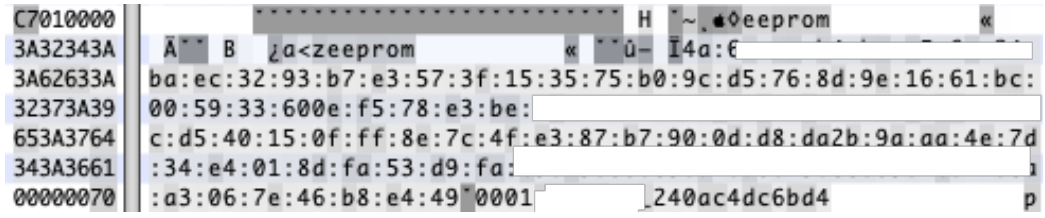

Figure 6.13: Plain-text private keys in EEPROM after encryption.

Fig. 6.14 shows the encrypted, unencrypted and unencryptable partitions to the Arduino minimal SPIFFS partition scheme. While it is possible to encrypt the EEPROM partition by setting the encryption flag, the Arduino EEPROM library does not support such operations as its uses esp_partition_write() to write data into the partition instead of the required esp_rom_spiflash_write_encrypted(). For NVS based EEPROM case, one

method to encrypt the private keys is to encrypt them in software before storing them into the NVS.

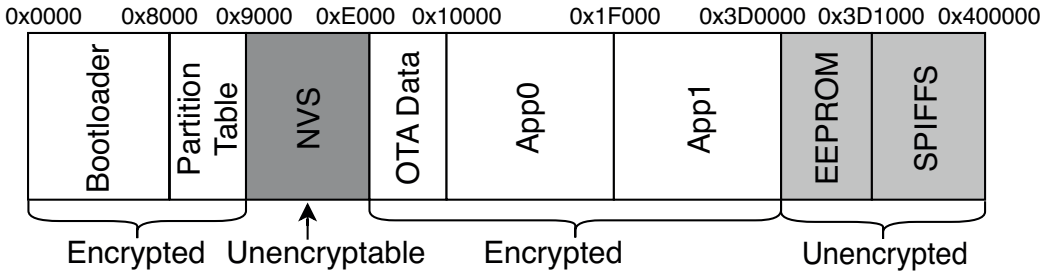

Figure 6.14: Encrypted and unencrypted partitions in ESP32 device.

This case study on secure configuration showed that even when one attempts to increase the security of a hardware/device via secure configuration, it may not guarantee its security in practice. Hence, in this thesis, we treat the secure configurations only as part of the solution of how to mitigate a threat, with further mechanisms being required to secure the system.

6.5 Summary

This chapter explored several case studies on secure system implementation. The case studies were implemented using the maker-oriented IoT hardware, MkIoT that was developed. An end-to-end (E2E) application prototype was implemented by leveraging existing off-the-shelf embedded hardware, open-source code, examples and tutorials provided by maker communities. Two case studies were carried out, where the first case study explored the implementation of secure end-to-end communication, and the second case study explored the implementation of secure configurations. The case studies highlighted issues when theoretically secure communication approaches may have their vulnerabilities, as well as issues with secure configuration, where the secure configuration may not be implemented correctly.

Chapter 7

Evaluation

In this chapter, the evaluation of the proposed automated secure system designer is presented. Three case studies are presented, where the first evaluates SecureWeaver from a secure corporate network design perspective, followed by a case study on secure IoT appliance system design. The final case study evaluates SecureWeaver for secure IoT hardware system design. The feature evaluation of SecureWeaver and its comparison with related works are also discussed in this chapter. The content of this chapter is a part of the publications [72, 73].

7.1 Case Study #1: Secure Corporate Network Design

In this section, the evaluation of SecureWeaver system is presented. First, an evaluation is performed from a functionality perspective, where SecureWeaver is utilized to generate a secure system design based on a set of example scenarios service requirement as input that includes several security threats. SecureWeaver will then verify that those threats are mitigated in the concrete system topologies outputted by SecureWeaver.

The set of example scenarios service requirements are based on a realistic corporate network scenario. Following the introduction of the input scenarios, the system design output is then evaluated from a security perspective to show that SecureWeaver is able to generate secure system designs.

7.1.1 Service Requirement Input for Evaluation

The experiments service requirement inputs are built using a subset of Weaver components and relationships that are also available in the SecureWeaver system model database, as shown in Fig. 7.1. The light-grey background in Fig. 7.1 denotes a group of abstract and concrete components that are related to each other. For example, `System` is an abstract component, whereas components such as `BackUpSys`, `WebSys`, and

ThinClientSys are the concrete representations of that system. The other included groups are abstract MiddleWare component, consisting of various concrete middleware software applications for backup, thin client and web application server, concrete WebApp component is related to web applications, and abstract Storage component, which has a Storage Area Network (SAN) system as a concrete component.

Fig. 7.1 also illustrates the other groups of system components: (i) Machine that consists of a hardware or a virtualized host; (ii) ExtThings that consists of external entities, such as users (User) and external API (ExtAPI); (iii) LAN that is related to Ethernet network switches; and (iv) the OS group, which consists of operating system-related components. Furthermore, Fig. 7.1 shows a number of independent components: (i) Wide Area Network (WAN); (ii) network router (Router); (iii) NIDS appliance (NIDS); (iv) thin client (ThinClient); (v) VPN server instance (VPNServer); and (vi) firewall appliance (Firewall). Fig. 7.1 also includes a component Requirement, which denotes functional or non-functional requirements for certain components that are part of a System in a topology.

Hence, a typical corporate network can be designed using the components in Fig. 7.1 that includes various network systems, such as thin clients, web application hosting, and remote access. In this evaluation we first introduce three basic scenarios, where each of the scenario has one specific MITRE ATT&CK threat explicitly defined. By carefully selecting the threats, we are able to demonstrate that SecureWeaver is able to evaluate all seven verification function with the selected threats. Moreover, the basic scenarios by themselves can be integrated with others to create more complex scenarios.

The scenarios below are introduced in the order of their complexity for designing them in SecureWeaver. To evaluate SecureWeaver's capability to design secure systems for real world applications, the complete service requirement that is build up from the three scenarios is shown in Fig. 7.2. Each of the building block scenarios are grouped using different shades of grey background and are numbered according to the description below.

7.1.1.1 Scenario #1

A service requirement with a thin client system is illustrated in Fig. 7.2 as Scenario #1. A thin client system and thin client components are defined, their component IDs being TCSys and TC, respectively. The connection between them is an abstract relationship, connTo, which is shown as a dashed line, and denotes that TC should have a path that connects it to TCSys in the output system design. The belongTo abstract relationship denotes that TC and TCSys belongs to bizLAN LAN network, where both TC and TCSys

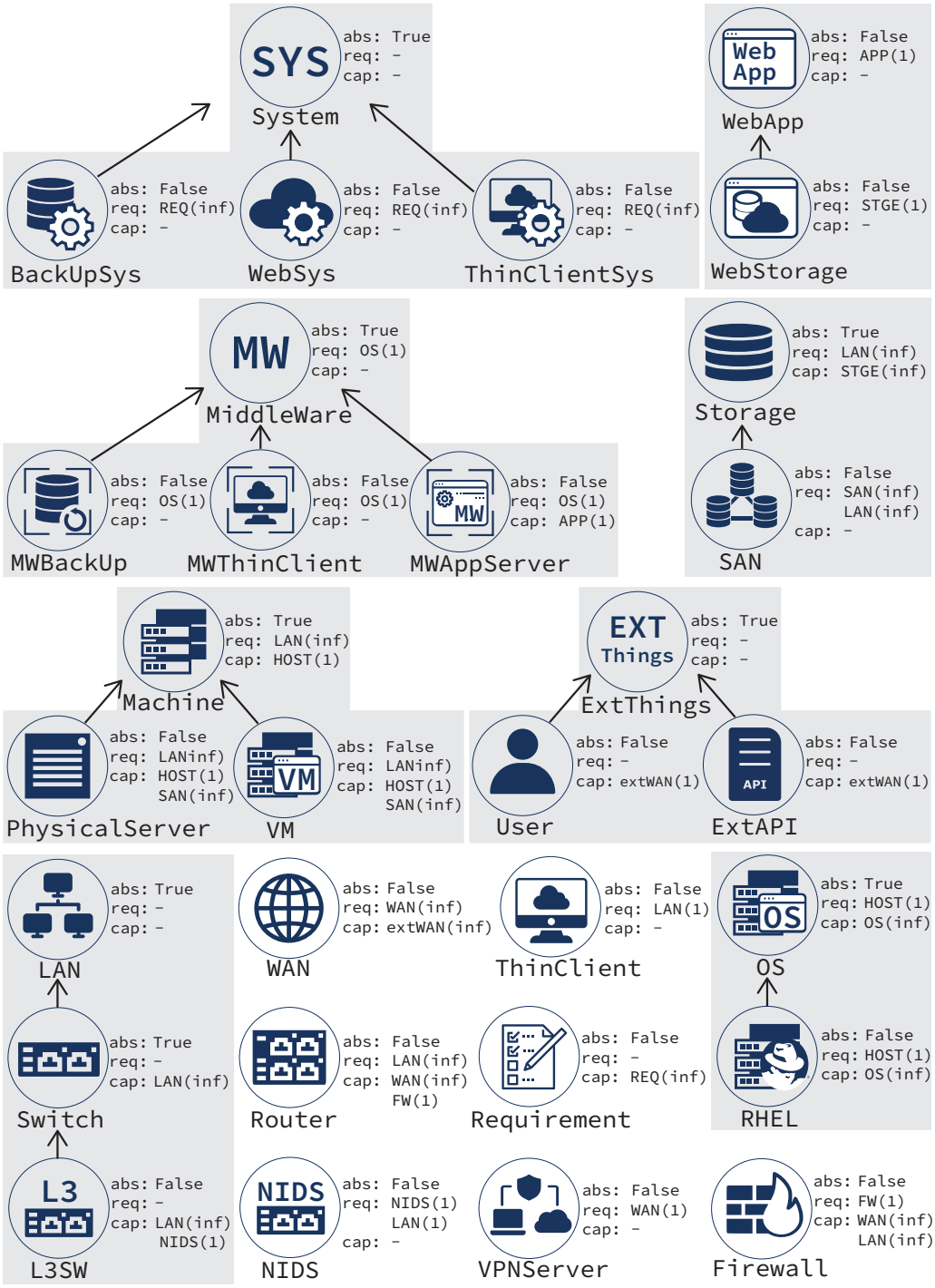

Figure 7.1: Properties and relationships of the components used in the system evaluation experiments.

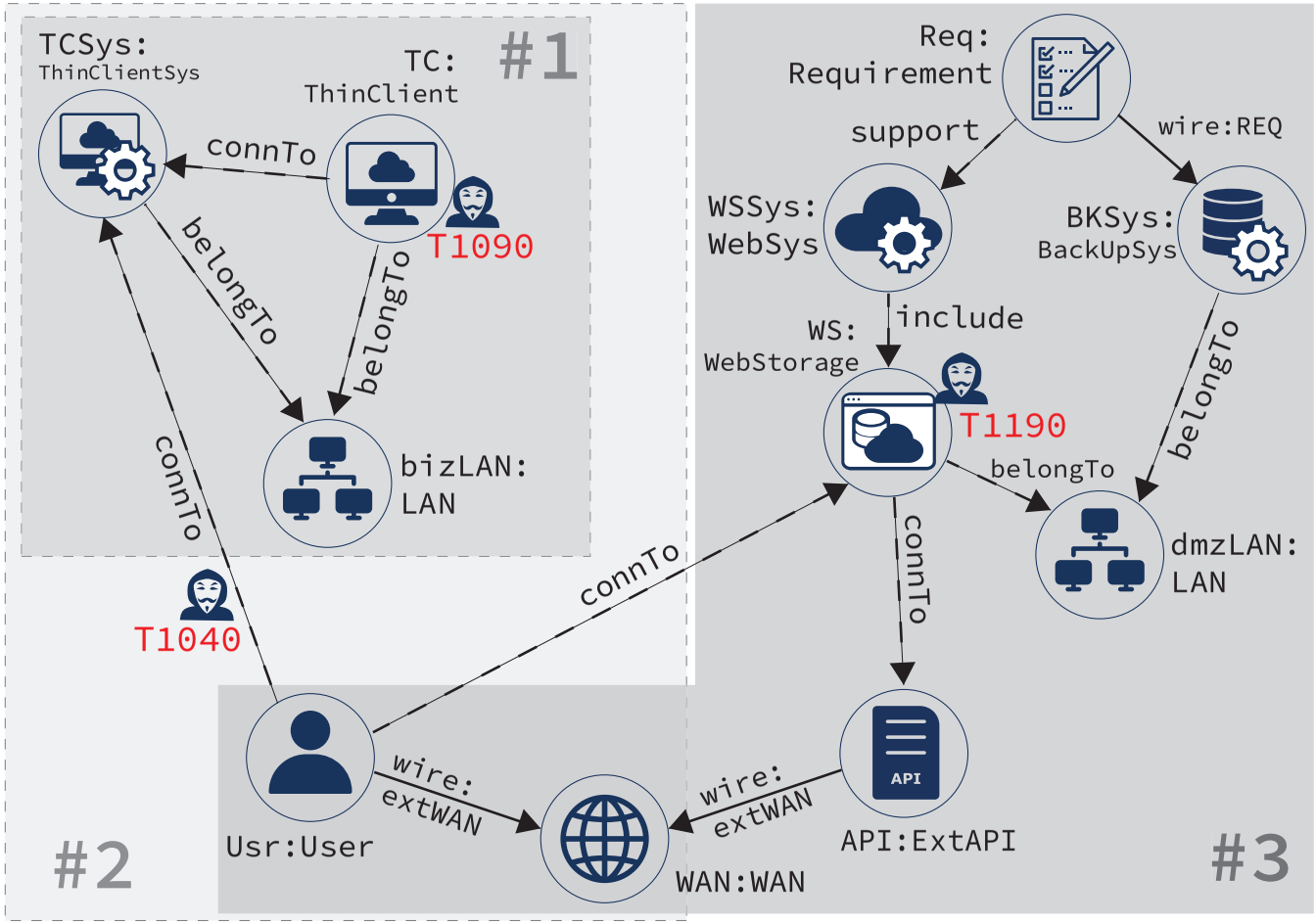

Figure 7.2: Input service requirement: thin client system with threats T1090 and T1040, and web system with threat T1190.

component or child component will be connected to `bizLAN` in the resulting system design. In this scenario the "Proxy" (T1090) threat is defined for the thin client component `TC` to model the case in which an adversary has control over `TC` and utilizes an external connection proxy as an intermediary to avoid suspicion over its command and control (C&C) communication traffic.

7.1.1.2 Scenario #2

Employees often need to remotely connect to a thin client environment on-premise to access software that is network licensed. This would require a user to connect to the company's thin client system via the public Internet, as illustrated in Scenario #2 in Fig. 7.2, where the `User` component is designated as `Usr` connected to `WAN` and `TCSys` by the `connTo` relationship. In this scenario the "Network Sniffing" (T1040) threat is defined for the `connTo` relationship, to model the case when the remote connection between `Usr` and `TCSys` is vulnerable to eavesdropping. The rest of the company's thin client system in the service requirement is the same with the Scenario #1 service requirement.

7.1.1.3 Scenario #3

Scenario #3 is a service requirement with a public-facing web application, as illustrated in Fig. 7.2. This scenario assumes a company has a on-premise private-cloud storage system, where its cloud storage is accessible via public Internet and utilized an external API for single sign-on (SSO) authentication. These are all illustrated in Fig. 7.2 as the abstract `connTo` relationship connecting the concrete `WebStorage` component together with both concrete external API (`API`) component and concrete external user (`Usr`) component, where both external components are connected to the `WAN`.

Furthermore, a backup system is introduced into the service requirement as the functional requirement that supports the private-cloud storage web system, `WSSys`. The `WSSys` is also connected via the abstract `include` relationship to its subsystem component, which is the web storage application `WS`. Both `WS` and the backup system component `BKSys` are part of the `dmzLAN` as connected with the abstract `belongTo` relationship. The MITRE ATT&CK Exploit Public-Facing Application (T1190) threat is explicitly defined for the `WebStorage` component in this scenario, to model the exploitation of zero-day vulnerabilities for public facing services such as `WS`.

7.1.1.4 Scenario #4

This scenario is a combination of the Scenario #1 and #3 service requirements, where both the thin client system and the private cloud storage system are assumed to exist together in the corporate network. Two threats are defined for this scenario, with threat T1090 affecting the thin client system, and threat T1190 affecting the web storage application.

7.1.1.5 Scenario #5

This scenario combines all the three basic scenarios, with the service requirement describing a real-life corporate environment that includes a thin client system, remote access, and a private-cloud storage system. This is illustrated in Fig. 7.2, with the thin client system being affected by the threat T1090, the remote access between the user and the thin client system being affected by the threat T1040, and the web storage application being affected by the threat T1190.

7.1.2 Evaluation Experiment Setup

SecureWeaver was implemented in Python 3 and is based on Weaver version 0.1.3. The source code of Weaver v0.1.3 was provided by NEC Corporation as a part of a joint research project with JAIST. An Amazon Web Services (AWS) Elastic Compute Cloud (EC2) VM instance was utilized to perform all the experiments presented in Sections 7.1.3 to 7.1.4. The EC2 VM instance that is utilized is "p2.xlarge", where the specifications are as shown:

- 4 virtual Intel Xeon E5-2686 v4 CPUs with a frequency of up to 3.0 GHz
- 64 GB of RAM
- 1 NVIDIA Tesla K80 GPU
- 60 GB of available storage

7.1.3 Security Verification Mechanism Evaluation

With the service requirement from the evaluation scenarios introduced in the previous subsection, a concrete and secure system design that corresponds to their service requirement can be generated by SecureWeaver. For the evaluation, a set of refinement rules that consists of 33 rules is utilized. This also includes the rules illustrated in Fig 4.3. Each available mitigation for a threat are assumed to be of equal effectiveness in mitigating the threat in

this evaluation. Hence, the total weight for security level is 1.0, being evenly divided among each mitigation for a threat.

Besides that, SecureWeaver has two modes of operation: (i) automatic; and (ii) interactive. In SecureWeaver's automatic mode, the first refinement rule that meets both quantitative and qualitative requirements will be matched by default, and heuristically applied them to generate the following potential topology states. This automatic refinement process is deterministic, only the first system design that satisfies the quantitative, qualitative and security requirements will be returned by SecureWeaver. On the other hand, SecureWeaver in interactive mode presents all valid refinement rules that are applicable to the current topology state, such that the user can manually choose the refinement rule to be applied at each refinement step.

The security level requirement for each scenario is set to the strictest standard, which is equivalent to 1.0, ensuring that the resulting system design passes every security verification function for the type of threat that is declared in the service requirement. Each scenario is individually evaluated with SecureWeaver in automatic mode and the resulting system design for the full scenario (Scenario #5) is shown in Fig. 7.3. As mentioned previously, the full scenario is a combination of Scenarios #1, #2 and #3, which are emphasized using different shades of grey background.

Scenario #1 The refined Scenario #1 part in Fig. 7.3 shows the thin client system (`TCSys`) that includes a thin client middleware (`TCS`), Red Hat Enterprise Linux (RHEL) operating system (`OS1`), and a physical server (`HOST1`) to host `TCS`. The thin client (`TC`) is logically connected to `TCSys` via `RDP` relationship and both thin client related components are physically connected to the same Layer 3 switch (`bizLAN`). Since `TC` is assumed to be affected by the threat T1090, it can be mitigated via M1037, M1031, and M1020, as referenced from the secure design database. Hence, security verifications such as traffic filtering verification (M1037), IDPS verification (M1031), and configuration settings verification (M1020) were performed by SecureWeaver.

The traffic filtering verification requires that the `TC` has either a software firewall or a hardware firewall, or both in the system design for ingress/egress networking filtering depending on the method specified in the threat mitigation knowledge base. The mitigation M1037 for threat T1090 specifies that traffic to known malicious network or infrastructure are to be blocked via the use of network allow and block list, which can be achieved via a software or a hardware-based firewall ("OR" method). Thus, `TC` security can be verified via the hardware firewall shown in Fig. 7.3. The IDPS verification

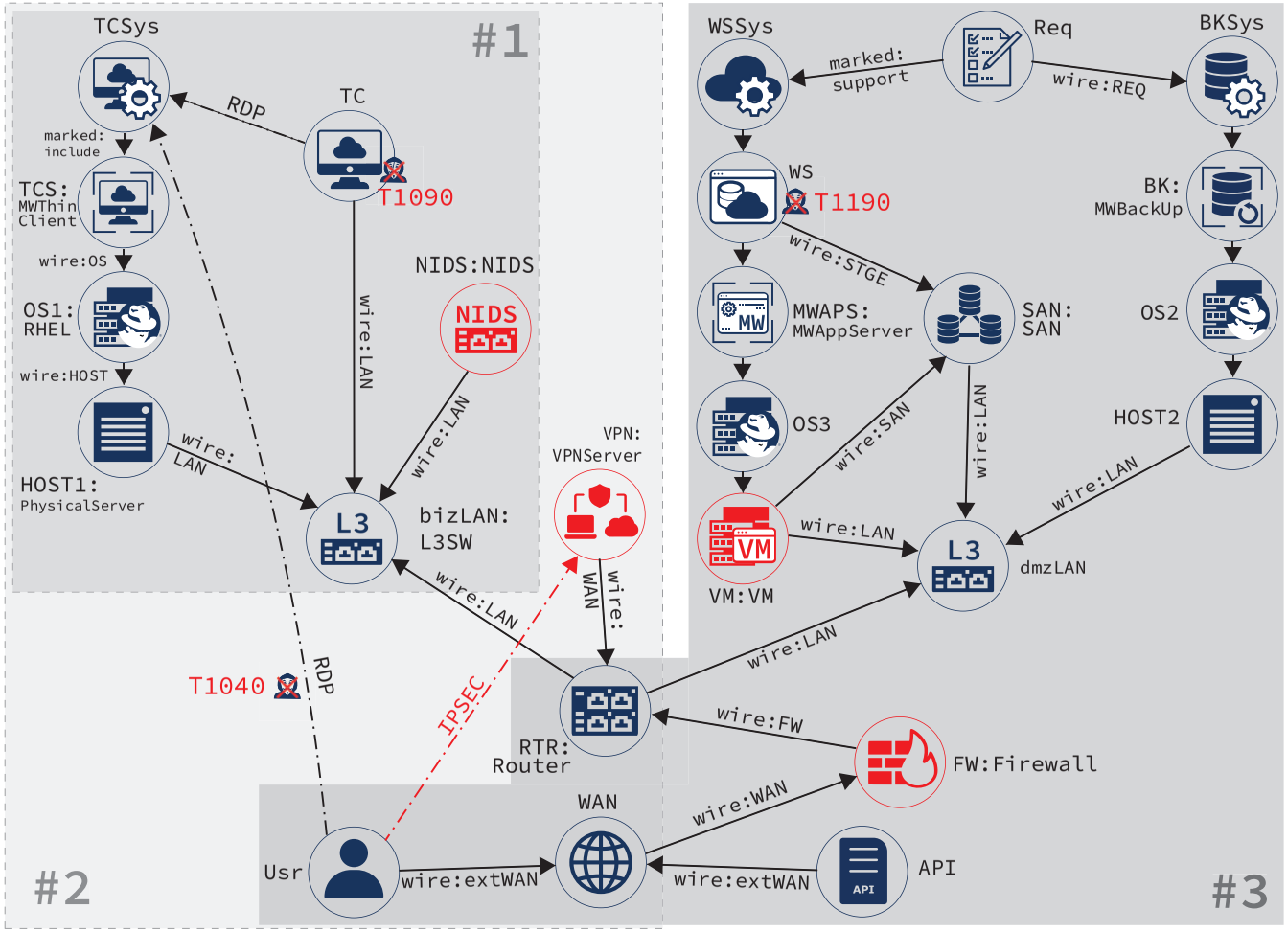

Figure 7.3: SecureWeaver output system design: refined thin client and web systems with mitigated threats.

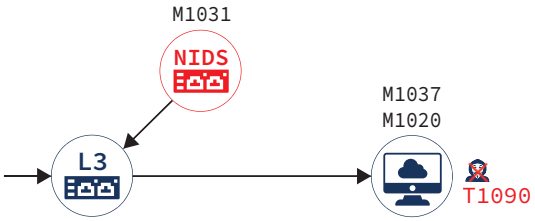

Figure 7.4: Scenario #1 attack path.

for mitigation M1031 requires the affected component local network to have an NIDS or IDPS component to monitor the network for suspicious traffics. The NIDS component (NIDS) highlighted in red is connected to the bizLAN network switch, thus satisfying the requirement. For configuration setting mitigations, the configuration settings verification verifies TC, for which it is assumed the mitigation M1020 will be implemented. Fig. 7.4 illustrates the attack path for Scenario #1, showing the entry of proxy attack through the switch, where the NIDS is connected to the switch as mitigation M1031, and configuration setting mitigations (M1037, M1020) assumed on the TC. As all mitigations against threat T1090 are in place, it is considered to be mitigated.

Scenario #2 For the refined Scenario #2 part in Fig. 7.3, SecureWeaver mitigates the threat T1040 on the remote access connection between the user and the thin client system. The secure protocol verification function is used in this case to verify the conceptual connection between Usr and TCSys. Since the remote desktop protocol (RDP) is not considered as a secure protocol in the threat mitigation knowledge base, the secure protocol verification validates the remote desktop connection security based on the IPSEC connection from the Usr to the VPN server (VPN) that is connected to RTR. Fig. 7.5 illustrates the attack path for Scenario #2, showing the entry of network sniffing attack between the Usr and TCSys. Since the starting point of the remote connection is at Usr, the network layer is encapsulated with an IPSEC tunnel to VPN for Usr to access the internal corporate network securely, which mitigates threat T1040.

Scenario #3 The refined Scenario #3 part in Fig. 7.3 includes the private cloud storage system and its supporting requirement (BKSys). The web storage application (WS) is affected by threat T1190, which is mitigated via M1048, M1050, M1030, and the configuration settings mitigations M1026, M1051, and M1016. For the application isolation and sandboxing verification (M1048), a virtual machine is required as a host, which is the VM in Fig. 7.3.

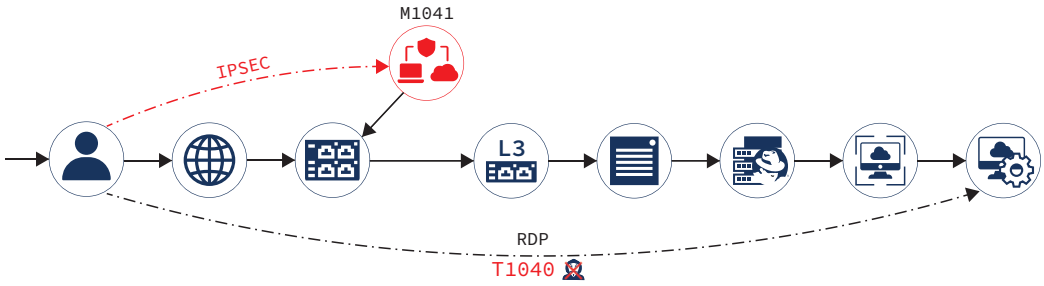

Figure 7.5: Scenario #2 attack path.

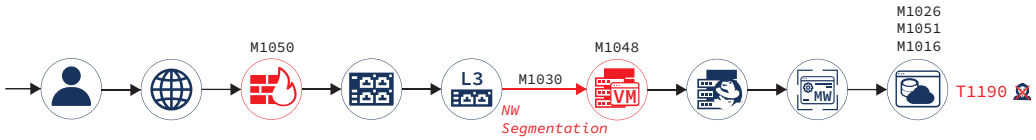

Figure 7.6: Scenario #3 attack path.

Moreover, the firewall (FW) that is placed between WAN and RTR satisfies the firewall use verification (M1050). The network segmentation verification is also successful as there are two LAN networks in Fig. 7.3, where the host of the WS affected by T1190, VM, is only connected to one of the LAN segments. The remaining mitigations M1026, M1051, and M1016 for WS are verified via configuration settings verification. Fig. 7.6 illustrates the attack path for Scenario #3, showing the entry of threat T1190 from Usr, where input traffics are first filtered by FW (mitigation M1050), and routed through the network that is only relevant to WS as a part of network segmentation (M1048). WS is hosted on VM, where it provides a sandboxed environment for the public-facing application (mitigation M1048), and lastly, mitigations related to configuration setting on WS (M1026, M1051, M1016) are assumed on the WS. As all mitigations against threat T1190 are in place, it is considered to be mitigated.

For the purpose of this evaluation, the discussion for Scenario #4 and #5 is not included, as they are just made of the first three scenarios. Table 7.1 summarizes the verification function(s) utilized by SecureWeaver for each evaluation scenario. Hence, we conclude that all three security threats in Fig. 7.3 are completely mitigated according to the security verification performed automatically by means of all the seven verification functions in SecureWeaver.

Table 7.1: SecureWeaver verification functions for each evaluation scenario.

Verification Function	Scenario				
	#1	#2	#3	#4	#5
Application Isolation and Sandboxing			◯	◯	◯
Firewall Use			◯	◯	◯
Networking Segmentation			◯	◯	◯
Configuration Settings	◯	◯	◯	◯	◯
Traffic Filtering	◯			◯	◯
Secure Protocol Use		◯			◯
Intrusion Detection and Prevention System Use	◯			◯	◯

7.1.4 Performance Evaluation

The performance characteristics of SecureWeaver and the overhead of the security verification mechanism is evaluated in this subsection. The performance evaluation is done in two approaches:

- Using SecureWeaver with increasingly complex input service requirements.
- Varying the number of security threats and the required security level for an in-depth analysis of complex system design feasibility and scalability

The evaluation is conducted using the five scenarios introduced in Section 7.1.1. The evaluation focuses on metrics such as the number of topology checks (iterations), and the time taken by various aspects of SecureWeaver (concretizing system design, security verification) needed to return a concrete and secure system design.

Five sets of experiments are performed for each scenario to obtain the average numerical results presented in the following subsections.

7.1.4.1 Security Verification Performance

The performance evaluation results of Scenario #1 to #5 are presented in Table 7.2, where S is the designation of the scenario number, n_{threat} is the number of service requirement threats, n_{topo} is the number of topology concreteness and quantitative checks, n_{sec} is the number of security verification iterations, $n_{sec,F}$ is the n_{topo} value for which the first security check is

performed, t_{total} is the total time in seconds to design and verify the system design, $RSD_{t_{total}}$ is the relative standard deviation of t_{total}, t_{sec} is the time in seconds for security verification, $RSD_{t_{sec}}$ is the relative standard deviation of t_{sec}, and t_{sec}/t_{total} is the percentage of t_{sec} with respect to t_{total}. Each of the five scenarios is evaluated both with a service requirement without any threat, and with a service requirement with threats with the maximum security level being defined (AND verification mode).

Beside the performance evaluation results in Table 7.2, the supplementary data is also included in Table 7.3 with statistics on the number of components and relationships, and the temporary database used by SecureWeaver to store data during the refinement and verification process. Thus, n_{comp} is the number of components in the system design, n_{rel} is the number of relationships in the system design, n_{all} is the total number of components and relationships in the system design, $size_{action}$ is the total size of the refinement action/step database, and $size_{state}$ is the total size of the topology state database.

When looking at the n_{topo} result for the service requirement without threat in Table 7.2, there is an increase with scenario complexity, and increase that is also reflected by the increasing number of n_{all} and size of $size_{state}$ in Table 7.3; actually, the $size_{state}$ is a more accurate indicator of the complexity of a scenario than n_{all}, as the database stores all the possible topology states in the search tree. The number of topology iterations for each scenario in both no security and maximum security level is shown in Fig. 7.7, plotted using logarithmic scale. A difference of 30.6% is observed between Scenario #1 and #2 for n_{topo} from Fig. 7.7, while there is a 19.1 times difference between Scenario #2 and #3. The combined Scenarios #4 and #5 show a drastic increase regarding n_{topo}, which illustrates the complexity of service requirements in real-world applications.

As for the additional number of n_{topo} iterations when using the full security level mode, the increase in percentage for Scenarios #1 to #5 is of 14.3%, 87.5%, 182%, 10.7%, and 7.2%, respectively. The increase in n_{topo} for the increasing scenario complexity is given by how Weaver heuristically refines for concrete topology states before security verification. The total n_{topo} is ultimately determined by the order of the possible topology states refined, where a topology state candidate that is both concrete and secure is first refined.

Nonetheless, the relative time taken by the security verification when computed as the ratio t_{sec}/t_{total} sharply decreases as the scenario becomes larger and more complex, given the intrinsic time needed for the basic design of such large scenarios. This shows that SecureWeaver security verification mechanism only introduces a small overhead; for example, t_{sec}/t_{total} for the

Table 7.2: SecureWeaver evaluation results: verification statistics and time measurements.

S	n_{threat}	n_{topo}		n_{sec}	$n_{sec,F}$	t_{total} [s]		$RSD_{t_{total}}$ [%]		t_{sec} [s]	$RSD_{t_{sec}}$ [%]	t_{sec}/t_{total} [%]
		w/o sec.	sec.			w/o sec.	sec.	w/o sec.	sec.			
1	1	49	56	5	49	0.204	0.278	0.26	0.39	0.00162	9.5	0.58
2	1	64	120	14	64	0.271	0.717	0.52	0.49	0.00405	8.5	0.56
3	1	1286	3626	18	3408	9.32	37.2	0.48	0.28	0.0103	3.6	0.028
4	2	47656	52754	122	47656	849	1123	0.19	1.7	0.0817	3.8	0.0073
5	3	199838	214152	266	199838	4622	5463	0.38	0.22	0.177	2.0	0.0032

Table 7.3: SecureWeaver evaluation results: topology statistics and disk data sizes.

S	n_{comp}		n_{rel}		n_{all}		$size_{action}$		$size_{state}$	
	w/o sec.	sec.	w/o sec.	sec.	w/o sec.	sec.	w/o sec.	sec.	w/o sec.	sec.
1	6	7	6	8	12	19	192 KiB	232 KiB	872 KiB	1.2 MiB
2	9	10	11	12	20	22	284 KiB	648 KiB	1.4 MiB	3.6 MiB
3	14	17	16	19	30	36	10 MiB	27 MiB	55 MiB	194 MiB
4	22	24	24	27	46	51	426 MiB	504 MiB	4.1 GiB	5.0 GiB
5	22	25	27	31	49	56	2.4 GiB	2.6 GiB	23.1 GiB	25.8 GiB

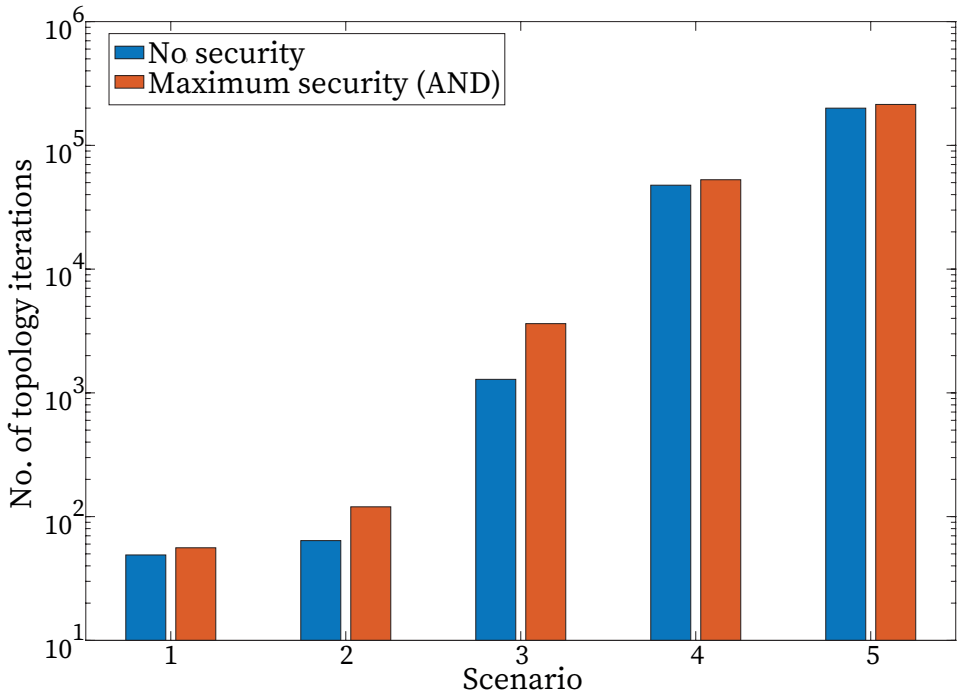

Figure 7.7: Number of topology iterations in each scenario (logarithmic scale).

full scenario in Fig. 7.3 is just 0.00032%, with an increase of 7.2% for n_{topo} due to the processing related to creating 265 concrete topology states that are discarded because they do not meet the security requirements.

Furthermore, the number of components and relationships in each scenarios for both no security and maximum security level are shown in Fig. 7.8 and Fig. 7.9. The additional components and relationships added to achieve maximum security system design is relatively minimal compared to the no security system design, with a maximum increase of three additional components for the most complex scenario and four additional relationships. This minimal increase of additional entities is due to the number of additional security-related refinement rules that is required in SecureWeaver. Table 7.4 summaries the number of security-related rules in SecureWeaver, where the total number of security-related rules is five. Only four mitigations (M1031, M1041, M1048, M1050) requires additional security-based components in their mitigation while the other mitigations rely on configuration setting in the relevant component.

For large and complex fully-abstract service requirements, SecureWeaver

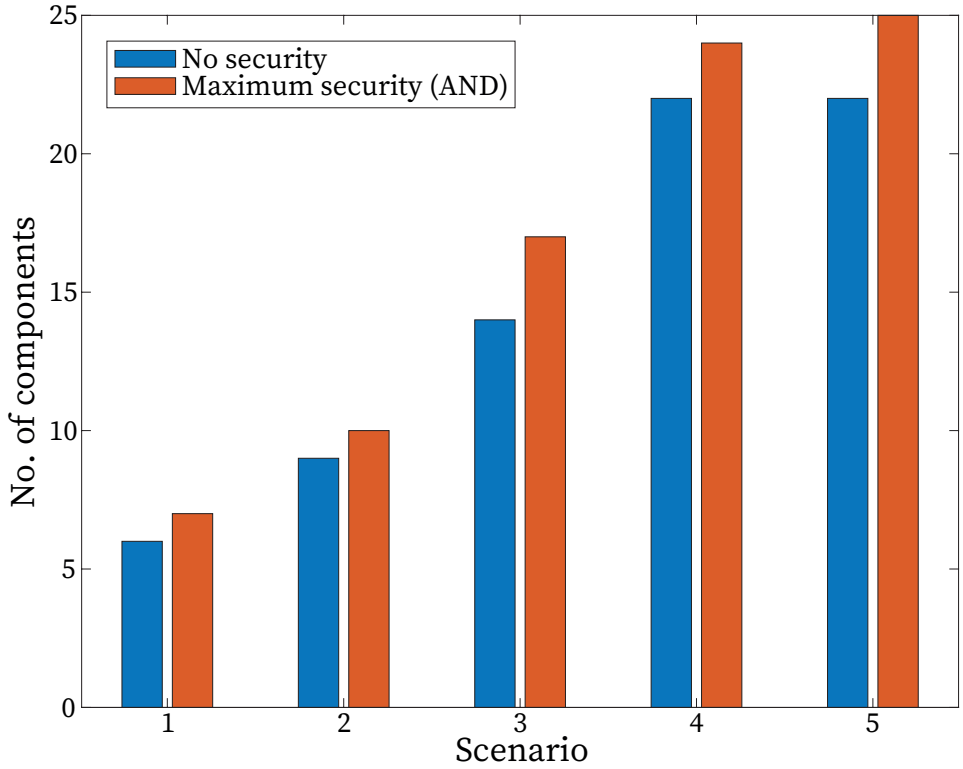

Figure 7.8: Number of components in each scenario.

is technically limited to the available storage space for storing temporary data (applied actions and computed topology states). As shown in Table 7.3, $size_{action}$ and $size_{state}$ increase with the complexity of the input service requirement, and in Scenario #5 the total size of temporary data reached 28.4 GB (considering that the total available storage on the experiment machine is 60 GB). However, in practice, networked systems are rarely designed from scratch. Hence, we also performed an evaluation of SecureWeaver in which an incremental addition of new services is done to an existing corporate network design. The resulting partially-concrete service requirement in this case included the abstract requirements in Scenario #1 and a concrete network topology of 205 entities (105 components and 100 relationships), and the output system design consisted of 110 components and 107 relationships, satisfying the imposed qualitative, quantitative and security requirements. In this way, the scale of the experiment was increased by about 4 times compared to the results reported in Table 7.3.

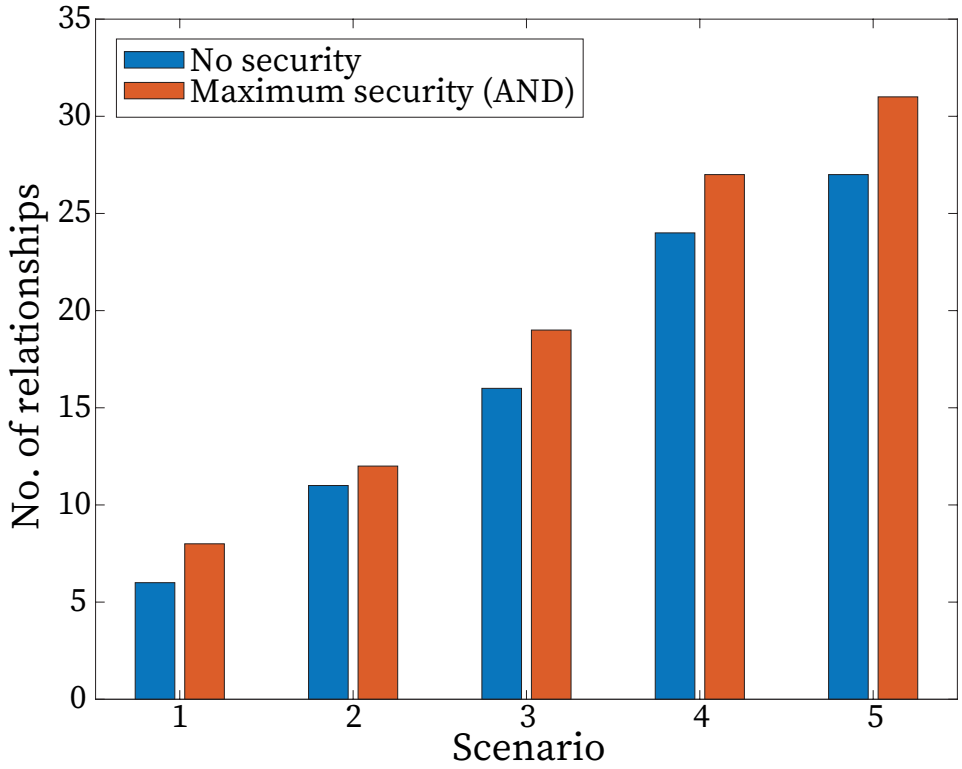

Figure 7.9: Number of relationships in each scenario.

7.1.4.2 Effect of Security Level on Performance

In real life networked system design, cost and other considerations may require the need for alternative system designs, so as to make possible trade-off analysis. When using SecureWeaver, one can generate multiple system designs that fulfill the functional requirements but have a different level of risk associated to them by adjusting the security level property of the threat in the service requirement (note that it is also possible to generate alternative solutions for the same level of risk).

In what follows we evaluate the effect on performance when varying the security level in Scenario #5 with a number of threats between one and three. In particular, the required security level is assigned the following values: no security (No Sec.), minimal security (OR), 25%, 50%, 75%, and full security (AND). The results for the second set of experiments are shown in Fig. 7.10 and Fig. 7.11, where both t_{total} and the ratio t_{sec}/t_{total} are plotted as function of the required security level shown on the horizontal axis.

The total elapsed time increases in general as the required security level

Table 7.4: Additional security-based refinement rules in SecureWeaver.

Scenario	Threat	Mitigation	Added Security Rules
1	T1090	M1037	0[1,2]
		M1031	1
		M1020	0[1]
2	T1040	M1041	2
		M1032	0[1]
3	T1190	M1048	1
		M1050	1
		M1030	0[1]
		M1026	0[1]
		M1051	0[1]
		M1016	0[1]

[1] Configuration-based mitigation
[2] May rely on existing security-based refinement rule(s)

becomes higher, although there are instances (e.g., the one threat scenario) when both security level 50% and 75% have similar t_{total}, since the topology state verified at the same number of n_{topo} is sufficiently secure, with the security level of the selected system design being larger than 75%.

As expected, the security verification overhead for minimum security level (OR) increases with the number of threats in the service requirement. Thus, for one threat mitigated at OR security level, an increase of 0.30% is recorded in t_{total} when compared with no security verification. For both two and three threats at OR security level, an increase of 8.3% and 9.3% is observed. Our results for n_{topo} at OR security level (results not shown in the thesis) show an increase of 0.0015% for one threat, and a 4.6% increase for both two and three threats when compared to the no security case.

The increase of t_{total} and t_{sec}/t_{total} observed in Fig. 7.10 and Fig. 7.11 as the security level requirement become higher is clear. For example, in the case of the maximum security level (AND), an increase of 4.0%, 12.6%, and 18.9% in t_{total} are recorded for a service requirement with one to three threats compared to the minimum security case (OR). Consequently, although a secure system design can be output by SecureWeaver in the shortest time by using a minimum security level requirement, a maximum security level solution can also be obtained in our experiments for a relatively small increase in time.

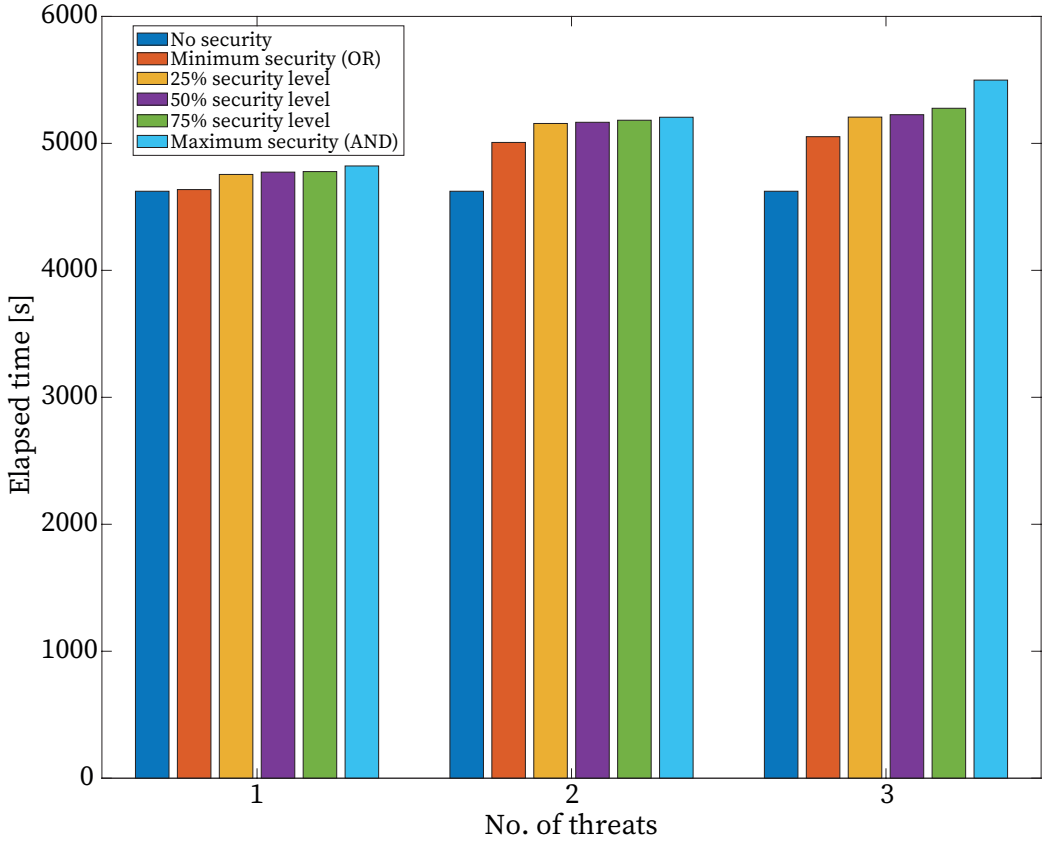

Figure 7.10: Number of threats versus the total elapsed time taken by SecureWeaver, with various security level requirements.

7.2 Case Study #2: Secure IoT Appliance System Design

In this section, a case study on IoT video surveillance service requirements is implemented. SecureWeaver is evaluated functionally by utilizing the IoT video surveillance service requirement to generate a secure system design that includes threats, and verify that they are mitigated in all the possible concrete topologies. Furthermore, a performance evaluation is conducted on SecureWeaver to determine its security verification overhead for scenarios with an increasing number of entities.

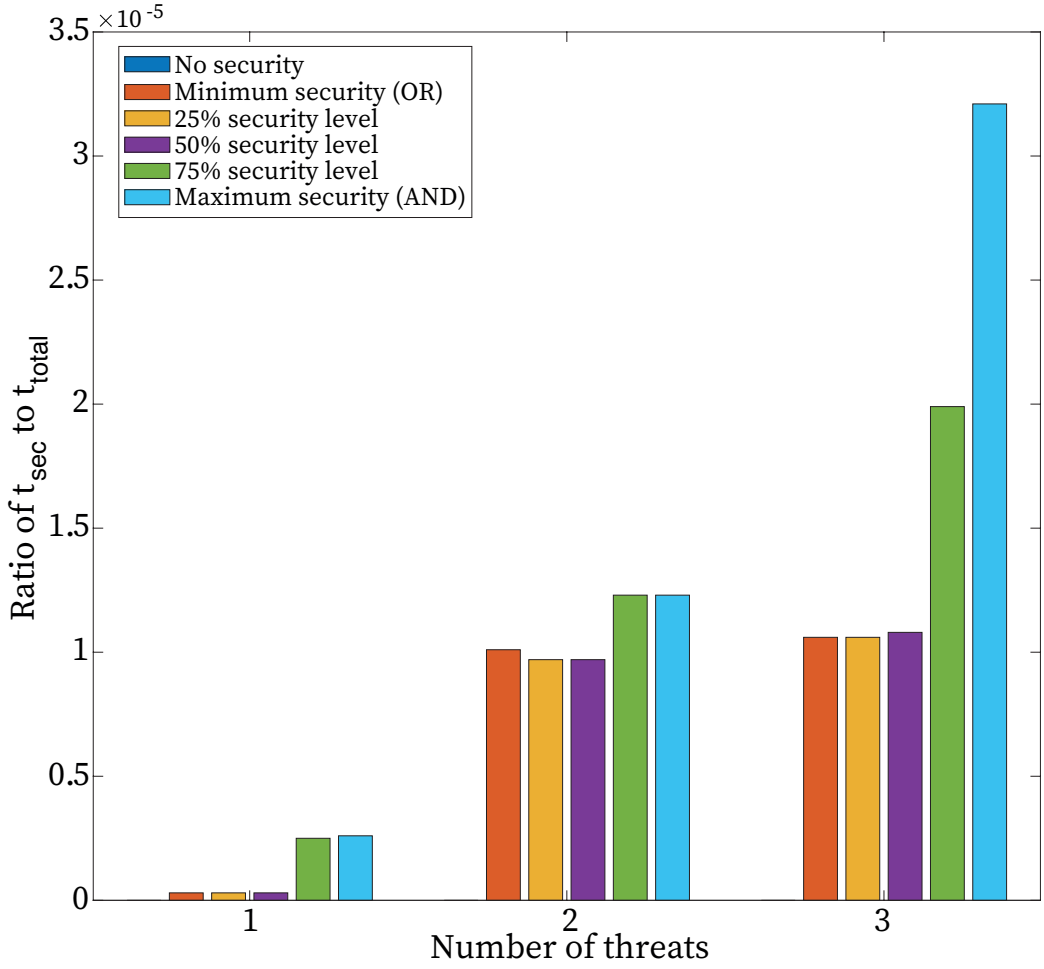

Figure 7.11: Number of threats versus ratio of t_{sec} to t_{total}, with various security level requirement.

7.2.1 Service Requirement Input for Evaluation

For this evaluation, Fig. 7.12 shows the following components that are used to build the service requirement, including all component types and their deriving relations. Both the video surveillance and health checker component types are derived from *App* component type while the component type's abstractness, *abs*, capability, *cap* and requirement, *req* are also shown in Fig. 7.12. An IoT IP camera, $IpCamera$ has $req = LAN(1)$ while a *Switch* type has $cap = LAN(inf)$, where LAN is an *rtype* and the number in the *rtype* corresponds to the minimum or maximum number that the same *rtype* can be "connected" to. This means that when an $IpCamera$ is defined in

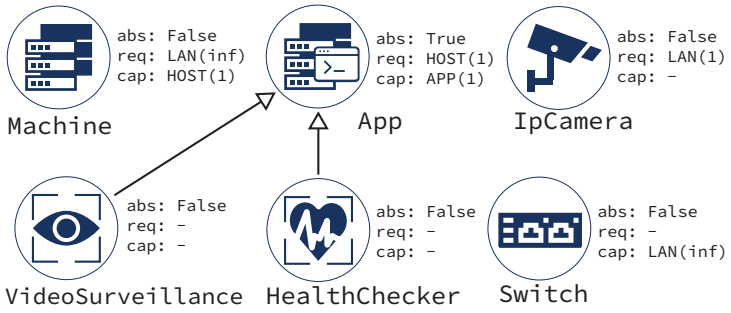

Figure 7.12: Properties and relationships of components for IoT appliance scenario.

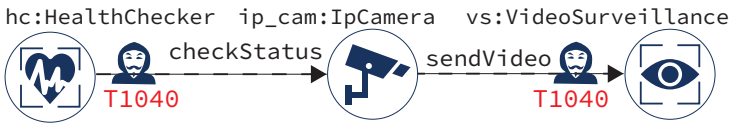

Figure 7.13: Service requirement input for IoT appliance scenario.

a service requirement, the final topology must have a maximum number of one *Switch* "connected" to it, while the *Switch* can be "connected" to an infinite number of components with $req = LAN$.

This service requirement assumes that a customer has an IP camera in the office and wants to sign up for a comprehensive security package that includes video surveillance and health checking services. The corresponding service requirement is shown in Fig. 7.13, where each icon represents a component type labeled with its *nid* and *ctype*. The component *ip_cam* of type *IpCamera* represents the existing IP camera in the customer's office, while the *hc* of type *HealthChecker* and *vs* of type *VideoSurveillance* are the services that are to-be deployed. Functional requirements of the new services are denoted as abstract relationships of type *checkStatus*, $e_{sendStatus}$, and *sendVideo*, $e_{sendVideo}$, where $e_{checkStatus}$ means *hc* has to regularly monitor *ip_cam* status and $e_{sendVideo}$ means *ip_cam* is required to stream its video feed to *vs*. The threats defined in *checkStatus* and *sendVideo* are both the MITRE ATT&CK-defined "Network Sniffing", T1040, indicating that there may be a threat to the confidentiality of the communication between the IP camera and its applications.

7.2.2 Evaluation Experiment Setup

The SecureWeaver experiments conducted in Sections 7.2.3 to 7.2.4.2 are perfosmed using an AWS EC2 VM instance, "t2.micro", which features one

virtual CPU with a frequency of up to 3.3 GHz and 1 GB of RAM. The "t2.micro" VM instance performs slightly faster than the "p2.xlarge" VM instance used in Chapter 7.1 due to its high boost CPU frequency. However, for sustained computation of complex system designs, the "t2.micro" VM instance is not a suitable candidate, as once it runs out of CPU credit, the VM will throttle to the base clock, affecting the performance evaluation results. Nevertheless, for the experiments conducted in this section, the "t2.micro" VM instance is sufficient.

7.2.3 Security Verification Mechanism Evaluation

After providing SecureWeaver with the evaluation service requirement, it generates a concrete and secure system design that corresponds to the input service requirement. Examples of possible concrete and secure scenarios are shown in Fig. 7.14, where the topology denoted with "1" in the top-left corner is the default system design output in automatic mode, and the other topologies are generated via interactive mode.

For the first system design in Fig. 7.14, *hc* and *vs* are communicating with *ip_cam* via inherently insecure application layer protocols ($HTTP$ and RTP), which do not mitigate the T1040 threats defined in the service requirement. Hence, SecureWeaver secures the topology by applied $IPSEC$ network protocol for both conceptual connections to mitigate their respective threats ($IPSEC$ mitigates T1040 as M1041, encrypt sensitive information). For system design 2, the communication between *hc* and *ip_cam* is replaced with $HTTPS$ in the application layer, which addresses T1040 via M1041; hence, the plain IP network layer protocol can be used for the conceptual connection between the *hc* and *ip_cam*. For system design 3, both *hc* and *vs* are communicating via $HTTPS$ and $SRTP$ on the application layer, both protocols mitigating T1040 via M1041. It is also possible to "oversecure" a conceptual connection, as shown in system design 4: $SRTP$ already mitigates the threat between *hc* and *ip_cam* at the application layer, but $IPSEC$ is used at the network layer.

Two examples of rejected topologies that are concrete but insecure are shown in Fig. 7.15. These topologies do not appear as the final system design, as they are rejected during the refinement and verification process. For the first rejected topology, both conceptual connection between *hc*, *vs* and *ip_cam* are insecure as both application and network layer protocol does not mitigates the threat. In the second rejected topology, while the conceptual connection between *vs* and *ip_cam* is secure, the topology is rejected due to the insecure conceptual connection between *hc* and *ip_cam*.

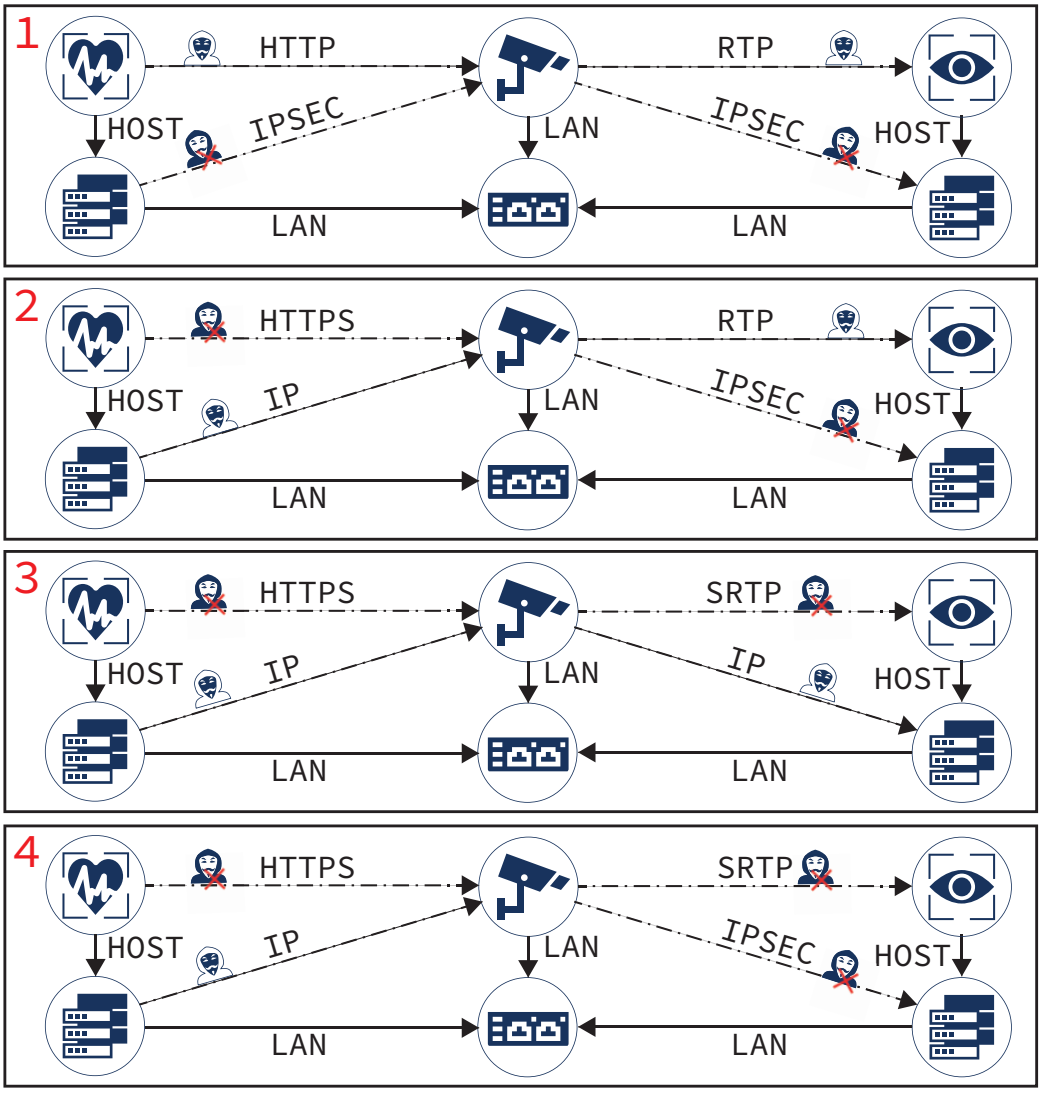

Figure 7.14: Examples of topologies for which the security verification was successful for the IoT appliance scenario.

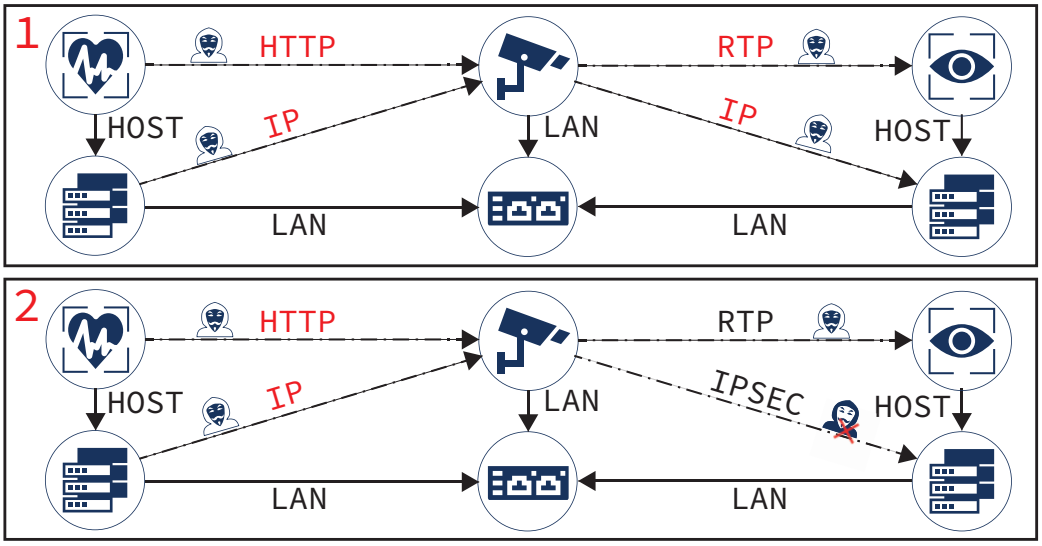

Figure 7.15: Examples of topologies rejected by the security verification algorithm for the IoT appliance scenario.

7.2.4 Performance Evaluation

In order to evaluate the overhead of the security verification mechanism, SecureWeaver is used to design a scenario with an increasing number of customer offices, n, ranging from 1 to 30. In this scenario, parameters such as time taken to design or verify a topology and the number of checks performed by portions of SecureWeaver to output a secure system design are evaluated. Since SecureWeaver performance evaluation is performed on an AWS EC2 "t2.micro" instance, the experiments are specifically done spaced in time to prevent exhausting the "t2.micro" instance burst CPU limits.

7.2.4.1 Expected System Design Output

The service requirement used is similar to that shown in Fig. 7.13, where each office contains a single IP camera and it is connected to a health checker and video surveillance application. When increasing the number of offices, the number of IP cameras also increase linearly along with the video surveillance applications as a pair. For the health checker application, there is only one instance, and all IP cameras will send their status to it. Hence, the expected system design output using SecureWeaver in automatic mode is shown in Fig. 7.16, where each office has an IP camera, a LAN switch and a video surveillance application, and the health checker application connects

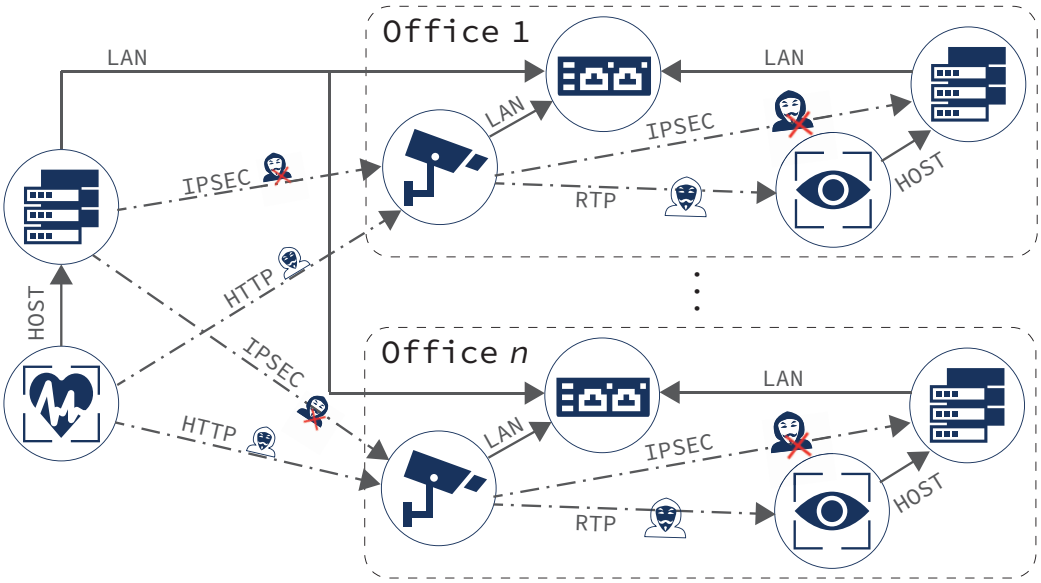

Figure 7.16: Expected output topology and security configuration for the IoT appliance scenario.

to every office's IP camera. Both $HTTP$ and $IPSEC$ are chosen by default in automatic mode as the application and network layer protocol in the their relationship, respectively.

7.2.4.2 Performance Evaluation Results

The performance evaluation results are shown in Table 7.5, where n is the number of offices, n_{comp} is the number of components in the system design, n_{rel} is the number of relationships in the system design, n_{all} is the total number of components and relationships in the system design, n_{threat} is the number of service requirement threats, n_{iter} is the number of algorithm iterations, n_{topo} is the number of topology concreteness and quantitative checks, n_{sec} is the number of security verifications,n_{child} is the number of component dependency searches, t_{total} is the total time elapsed in seconds to design and verify the system design, $RSD_{t_{total}}$ is the relative standard deviation of t_{total}, t_{sec} is the time elapsed in seconds for security verification, $RSD_{t_{sec}}$ is the relative standard deviation of t_{sec}, and t_{sec}/t_{total} is the percentage of t_{sec} from the total t_{total}.

The results in Table 7.5 are the averages of 10 experiments for each number of offices n. As both n_{comp} and n_{rel} increase, the average t_{total} increased in a manner that can be curved fitted ($R^2 = 1$) as shown in

Table 7.5: Numerical results for the performance evaluation of SecureWeaver.

n	n_{comp}	n_{rel}	n_{all}	n_{threat}	n_{iter}	n_{topo}	n_{sec}	n_{child}	$t_{total}[s]$	$RSD_{t_{total}}[\%]$	$t_{sec}[s]$	$RSD_{t_{sec}}[\%]$	$t_{sec}/t_{total}[\%]$
1	6	8	14	2	25	44	9	1	0.13	1.02	0.0021	1.73	1.57
2	10	16	26	4	101	220	21	11	0.87	0.69	0.0045	4.37	0.51
3	14	24	38	6	190	499	37	27	2.57	0.90	0.0073	1.24	0.29
4	18	32	50	8	265	761	57	49	5.14	0.69	0.012	0.92	0.23
5	22	40	62	10	335	995	81	77	8.80	0.59	0.014	2.08	0.16
6	26	48	74	12	405	1222	109	111	13.87	0.43	0.016	1.25	0.12
7	30	56	86	14	475	1449	141	151	20.89	3.35	0.019	1.85	0.09
8	34	64	98	16	545	1676	177	197	28.99	0.47	0.023	1.76	0.08
9	38	72	110	18	615	1903	217	249	39.61	0.64	0.026	3.76	0.07
10	42	80	122	20	685	2130	261	307	52.41	0.78	0.032	9.46	0.06
15	62	120	182	30	1035	3265	541	687	159.04	0.39	0.061	7.00	0.04
20	82	160	242	40	1385	4400	921	1217	359.97	0.52	0.12	6.58	0.03
25	102	200	302	50	1735	5535	1401	1897	689.97	0.46	0.21	15.50	0.03
30	122	240	362	60	2085	6670	1981	2727	1195.69	0.48	0.31	13.65	0.03

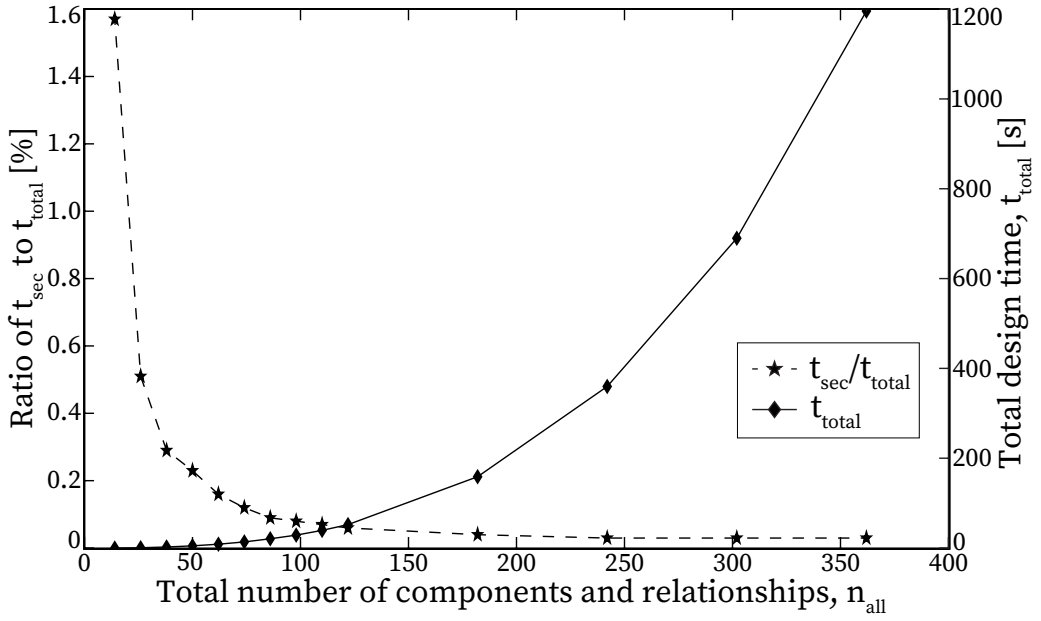

Figure 7.17: Performance evaluation results for SecureWeaver for the IoT appliance scenario.

Equation 7.1, where n_{all} is the sum of n_{comp} and n_{rel}:

$$t_{total} = 3 \cdot 10^{-5} \cdot n_{all}^3 - 0.001 \cdot n_{all}^2 + 0.1834 \cdot n_{all} - 3.9823. \tag{7.1}$$

The average t_{sec} to the total number of n_{comp} and n_{rel} can be curved fitted ($R^2 = 0.9992$) as shown in Equation 7.2:

$$t_{sec} = 4 \cdot 10^{-9} \cdot n_{all}^3 + 8 \cdot 10^{-7} \cdot n_{all}^2 + 8 \cdot 10^{-5} \cdot n_{all} - 0.003. \tag{7.2}$$

A more detailed analysis shows that the above performance is an intrinsic characteristic of the original Weaver. When considering only the security verification part of the system, it is noted that the average elapsed time, t_{sec}, increases much slower. Thus, when expressed in percentages, t_{sec} versus t_{total} decreases sharply from 1.57% for $n = 1$ to 0.03% for $n = 30$. These results are plotted and shown in Fig. 7.17. For real-world system designs have more than 100 entities (9 to 30 offices), the overheads are 0.07% at $n = 9$ to 0.03% at $n = 30$ with an average overhead of 0.04%. This means that the SecureWeaver extension only incurs a small overhead, which becomes comparably very small with respect to the total processing time as n_{comp} and n_{rel} increase.

7.3 Case Study #3: Secure IoT Hardware System Design

In the previous section, a case study on an IoT video surveillance appliance was conducted at a network system design level. In this section, a case study is conducted to explore the feasibility of performing hardware level system design for IoT with SecureWeaver. First, the formal representation of each of the intrinsic components that make up a typical IoT hardware device is defined. This is followed by the details on Message Queueing Telemetry Transport (MQTT) as the IoT communication in end-to-end system design. The corresponding refinement rules for IoT system design are presented, and functional evaluations are carried out to validate the feasibility of SecureWeaver in both designing concrete hardware level system design and ensuring the design is free of security threat.

7.3.1 Formalization of IoT Components in SecureWeaver

To formalize an IoT components for SecureWeaver, an IoT application or device with a specific application running on an embedded device, which is connected to the network for communication to an endpoint, is assumed. The IoT application or device can be broken down into smaller components, not restricted to the following:

- IoT embedded application
- Operating system of the device
- Network interface of the device

The embedded application or firmware of an IoT device is analogous to an IT web application or software application. In SecureWeaver, such firmware can be defined in a JSON structure as shown in Code 7.1, where the IoT application is a concrete component type, where the amount of memory ("memory_use") can be defined as an integer. The "reference" key denotes a requirement for the `IoT_App` that it is required to be hosted on one and only one OS component ("iot_os"). The amount of memory used can be defined in the intent, where SecureWeaver has to search and match relevant rule candidates to meet the quantitative requirement.

Code 7.1: Description of IoT firmware in SecureWeaver

```
1 "IoT_App": {
2     "abstract": false,
3     "derived_from": [],
4     "properties": {
```

```
5           "memory_use": "Integer"
6       },
7       "reference": {
8           "iot_os": 1
9       }
10  }
```

An IoT OS is analogous to an IT OS, which can be directly described in SecureWeaver as an abstract OS component as shown in Code 7.2. This generic OS (`IoT_OS`) has a "service" key, which denotes that the generic OS provides the "iot_os" capability, where component such as `IoT_App` can be connected to that it satisfies the `IoT_App` requirement. The "inf" value on the "service" key denotes that the abstract OS is able to provide an unlimited number of "iot_os" service, as long as the other quantitative requirements are met (e.g., "memory_use").

Code 7.2: Description of abstract IoT OS in SecureWeaver

```
1   "IoT_OS": {
2       "abstract": true,
3       "derived_from": [],
4       "service": {
5           "iot_os": "inf"
6       },
7       "reference": {
8           "iot_hardware": 1
9       }
10  }
```

The abstract IoT OS component has to be refined into a concrete component in SecureWeaver. Hence, the FreeRTOS real-time OS (RTOS) for embedded system is defined as shown in Code 7.3, where the `IoT_FreeRTOS` is derived from the `IoT_OS` component describing its inheritance. The abstract value is defined as False, with the capability to provide unlimited "iot_os". Its service requirement requires one "iot_hardware", which the OS is required to be hosted on a hardware platform.

Code 7.3: Description of concrete FreeRTOS OS in SecureWeaver

```
1   "IoT_FreeRTOS": {
2       "abstract": false,
3       "derived_from": ["IoT_OS"],
4       "service": {
5           "iot_os": "inf"
6       },
7       "reference": {
8           "iot_hardware": 1
9       }
```

```
}
```

An abstract IoT hardware (`IoT_HardwarePlatform`) can be formalized in SecureWeaver as shown in Code 7.4, where it provides one "iot_hardware" as its capability and requires at least one to an unlimited number of abstract network interface ("iot_network"). For the concrete hardware platform component, the Espressif ESP32 is used as the example, where its capabilities are defined as a SecureWeaver component. This is shown in Code 7.5, where the `IoT_ESP32` abstractness is False and it is derived from the `IoT_HardwarePlatform`. The `IoT_ESP32` also provide capabilities such as "iot_hardware","iot_wifi", "iot_ble", "iot_i2c", "iot_uart", "iot_spi", and "iot_crypt_engine". An interface service, such as "iot_wifi", provides the option for SecureWeaver to determine whether a certain precursor component requires such capability. For example, if an IoT application require connection to an I2C capable temperature sensor, the component that is matched by SecureWeaver should have "iot_i2c" as a part of its capability.

Code 7.4: Description of abstract hardware platform in SecureWeaver

```
"IoT_HardwarePlatform": {
    "abstract": true,
    "derived_from": [],
    "service": {
        "iot_hardware": 1
    },
    "reference": {
        "iot_network": "inf"
    }
}
```

Code 7.5: Description of concrete ESP32 based hardware platform in SecureWeaver

```
"IoT_ESP32": {
    "abstract": false,
    "derived_from": ["IoT_HardwarePlatform"],
    "service": {
        "iot_hardware": 1,
        "iot_wifi": 1,
        "iot_ble": 1,
        "iot_i2c": 1,
        "iot_uart": 1,
        "iot_spi": 1,
        "iot_crypt_engine": 1
    },
    "reference": {
        "iot_network": "inf"
```

```
15      }
16  }
```

The last component of the IoT application/device is the network interface. The abstract network interface is defined in SecureWeaver as shown in Code 7.6, where it provides an unlimited number of "iot_network" capability and requires at least one LAN ("lan") service. The LAN and the rest of the components are generic SecureWeaver IT/NW components, such as the network switch, router, WAN, external services, etc. A concrete network interface, `IoT_Network_WiFi` is shown in Code 7.7, where it provides one "iot_network" capability.

Code 7.6: Description of abstract network interface in SecureWeaver

```
1  "IoT_NetworkInterface": {
2      "abstract": true,
3      "derived_from": [],
4      "service": {
5          "iot_network": "inf"
6      },
7      "reference": {
8          "lan": "inf"
9      }
10 }
```

Code 7.7: Description of concrete WiFi network interface in SecureWeaver

```
1  "IoT_Network_WiFi": {
2      "abstract": false,
3      "derived_from": ["IoT_NetworkInterface"],
4      "service": {
5          "iot_network": 1
6      },
7      "reference": {
8          "lan": "inf"
9      }
10 }
```

7.3.2 Refinement Rules for IoT System Design

With the basic concepts that were described in the previous two subsections, the refinement rules can be defined for SecureWeaver to be able to design IoT system. All the refinement rules that are used in this case study are listed in the Appendix A under "iot_rule.json".

Modeling IoT Hardware For each of the components introduced in Section 7.3.1, their corresponding refinement rules and their brief descriptions are summarised as follows:

1. `IoT_OS` → `IoT_FreeRTOS` (abstract → concrete): Refine an abstract IoT OS component into a concrete FreeRTOS component.
2. `IoT_HardwarePlatform` → `IoT_ESP32` (abstract → concrete): Refine an abstract embedded hardware component into a concrete ESP32 one.
3. `IoT_NetworkInterface` → `IoT_Network_WiFi` (abstract → concrete): Refine an abstract embedded network interface component into a concrete WiFi interface component.
4. `IoT_App` → `IoT_OS`: Add an abstract embedded OS component to host the `IoT_App` component.
5. `IoT_OS` → `IoT_HardwarePlatform`: Add an abstract embedded hardware component to host the `IoT_OS` component.
6. `IoT_HardwarePlatform` → `IoT_NetworkInterface`: Add an abstract embedded hardware network interface component to the `IoT_HardwarePlatform` component.
7. `IoT_NetworkInterface` → `Switch`: Add an abstract network switch component to the `IoT_NetworkInterface` component.

IoT Communication via Message Queueing Telemetry Transport (MQTT) The refinement rules presented above are adequate to refine an `IoT_App` component into a standalone IoT appliance that connects to a LAN network (WiFi access point is equated to a network switch). The case study presented in Section 6.3 is revisited, where we assumed the scenario of the IoT application communicating with an endpoint such as an external API. Such, the refinement rules for connecting the `IoT_App` component to the External API component are based on the Message Queueing Telemetry Transport (MQTT) protocol over IP network.

By default, the original MQTT protocol does not require authentication as a mandatory requirement [100]. The MQTT protocol does have basic authentication, which is illustrated in Fig. 7.18, where the MQTT client provides the user ID and password as credential to the MQTT broker, for which the MQTT broker will authenticate and acknowledge whether the authentication is successful. However, the basic authentication method is insecure, as the credentials are sent as plain text over the network. This sensitive information can be intercepted over the network via security threat such as MITRE ATT&CK defined Network Sniffing (T1040).

Since both no-authentication and basic authentication methods of MQTT are unsecure, to demonstrate SecureWeaver capability of designing a secure

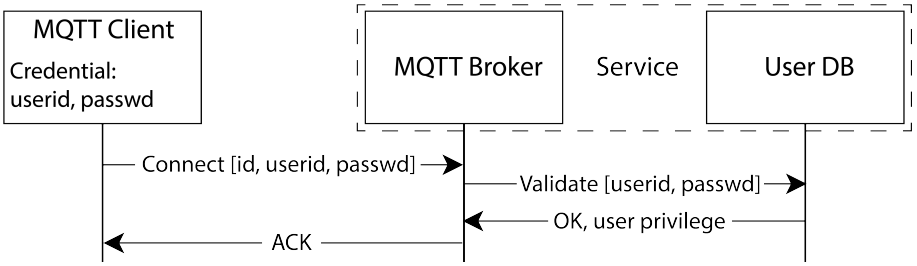

Figure 7.18: MQTT communication using basic authentication.

system design, a secure method of MQTT authentication is illustrated in Fig. 7.19. The MQTT communication is encrypted via a TLS tunnel, where the credential in plain text can be transmitted securely to the MQTT broker for authentication.

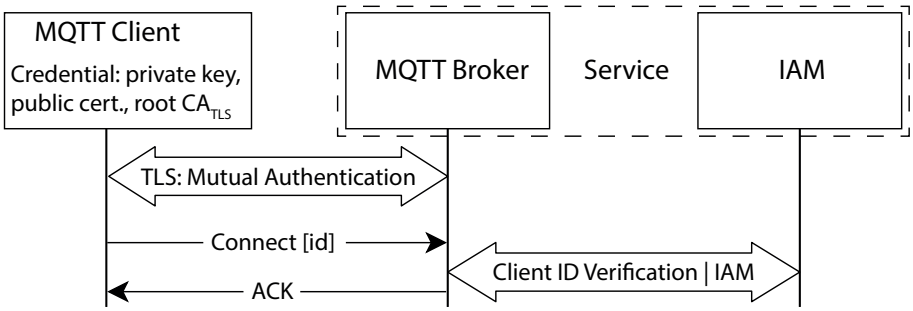

Figure 7.19: MQTT communication using TLS.

Modeling MQTT Communication Two possible end-to-end MQTT communication scenarios between an IoT appliance and an endpoint over an IP network can be created, where one utilizes an insecure approach and the other one uses a secure approach based on TLS to communicate with the endpoint. The refinement rules for these two MQTT communication approaches are summarized below:

1. `IoT_App` → external API (MQTT only): Refine abstract `connTo` relationship to a pair of concrete MQTT and IP logical connections
2. `IoT_App` → external API (MQTT with TLS): Refine an abstract `connTo` relationship to a pair of concrete MQTT-TLS and IP logical connections

The refinement rule for secure end-to-end communication of `IoT_App` with external API is illustrated in Fig. 7.20, where the abstract `connTo`

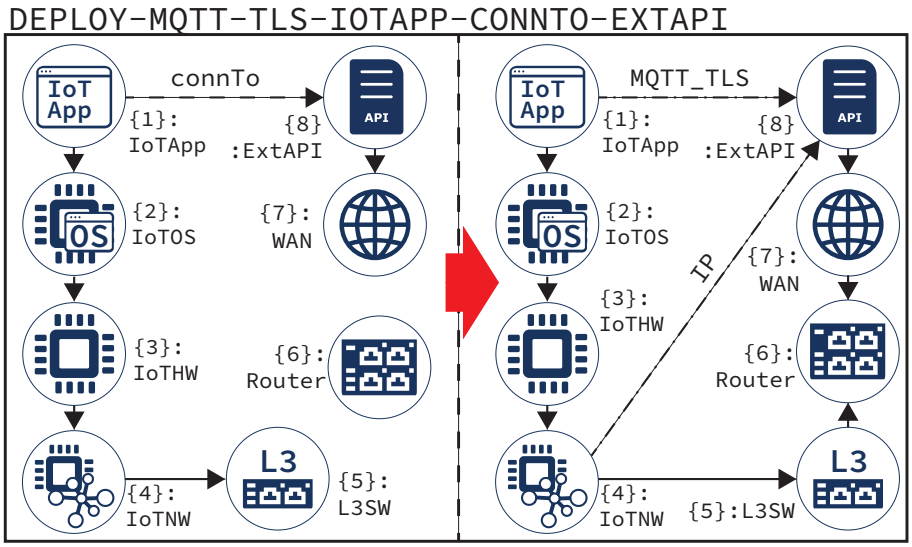

Figure 7.20: Graphical representation of secure MQTT refinement rules.

relationship between `IoTApp` and `ExtAPI` and the other relevant components on the left-hand side are transformed into the right-hand side topology in a one-step refinement.

Considering both the IoT hardware and communication aspects discussed in this section, a total of nine IoT-specific refinement rules were added to the SecureWeaver model database in order to add to it the capability of designing secure IoT systems.

7.3.3 Secure IoT System Design Evaluation

For the evaluation, the same SecureWeaver version 0.1.3 setup on AWS "t2.micro" VM instance detailed in Section 7.2.2 was utilized. Three scenarios are designed to evaluate SecureWeaver for hardware-level system design:

1. Standalone IoT system design
2. End-to-end IoT application without security
3. End-to-end IoT application with one security threat

The standalone IoT system design is a basic IoT application that is used to functionally evaluate SecureWeaver capability at designing hardware level system. The end-to-end IoT application (with no-security scenario) is the extension of the first evaluation scenario, where SecureWeaver designs at both hardware-level and network-level. The end-to-end IoT application scenario

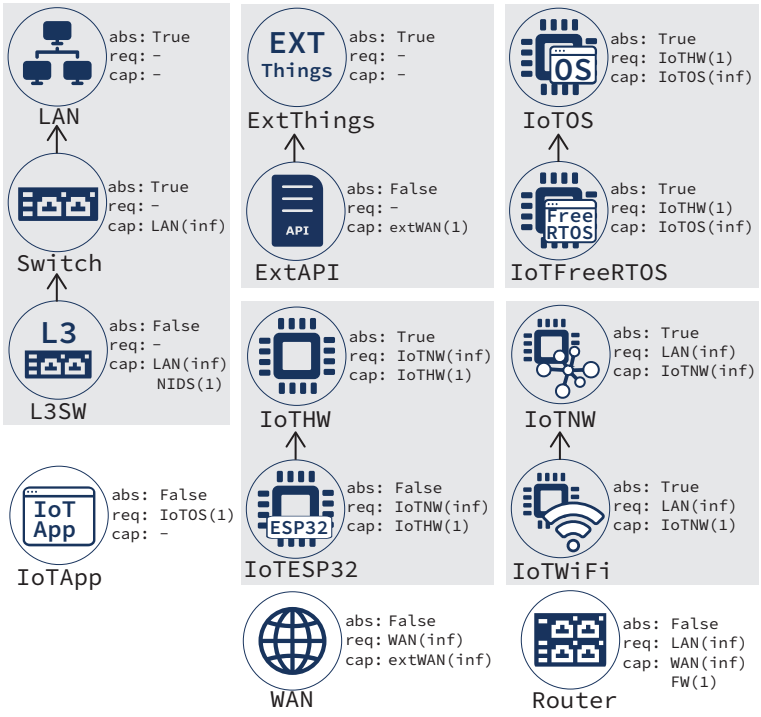

Figure 7.21: IoT-related and supporting components in SecureWeaver used in the evaluation.

with security threat on the other hand is used to evaluated SecureWeaver capability to mitigate security threats for IoT system design.

The components that are used in the evaluation are shown in Fig. 7.21. The grey background denotes the grouping of related type components, for example the abstract LAN component is the root for both abstract Switch component and the concrete layer 3 network switch, L3SW.

7.3.3.1 Evaluation of Standalone IoT System Design

The service requirement for standalone IoT system design evaluation is illustrated in Fig. 7.22, where it defines a concrete IoT application, `App1` that is an "IoTApp" component type introduced in the previous subsection. This single IoT application is used to evaluate SecureWeaver capability to refine the application from the application level to the networking level.

The result of SecureWeaver for the standalone IoT system design scenario is shown in Fig. 7.22. The `IoTApp-1` is refined where a concrete OS, `IoT_FreeRTOS` provides the "iot_os" capability that is required. The `IoT_ESP32` provides the "iot_hardware" capability to `IoT_FreeRTOS` hard-

IoTApp-1:IoTApp

Figure 7.22: Service requirement of standalone IoT system.

ware platform while the IoT_Network_WiFi provides the network capability required by the hardware platform to connect to a network. Each of their relationships are shown in the "Topology" section in Fig. 7.22 under "Edge", where the first and second column is the source and destination, while the third column is the type of the relationship.

The resulting system design for Scenario #1 is represented graphically in Fig. 7.24, where the IoTApp-1 component is connected via wire:IoTOS to the concrete OS component IoT_FreeRTOS that provides the iot_os capability that is required by IoTApp-1. The IoT_ESP32 component provides the iot_hardware capability to IoT_FreeRTOS hardware platform, while the IoT_Network_WiFi component provides the network capability required by IoT_ESP32 to connect to a network. The IoT_Network_WiFi is finally connected to a network switch, L3SW (this generic L3SW component represents a WiFi access point). From the output design we conclude that SecureWeaver is able to functionally design an IoT system automatically.

7.3.3.2 Evaluation of End-to-End IoT Application Without Threats

In this scenario, the basic standalone IoT application intent is further extended, and the IoT application, IoTApp-1 is connected to an external API, External_API_1 as described by an abstract connTo relationship as shown in Fig. 7.25. Besides that, External_API_1 is also connected to the WAN via a concrete relationship, wire:external-GW. Hence, this scenario describes an end-to-end communication between the IoT application and the external API, which can be a cloud platform on the Internet.

The result of the scenario refined by SecureWeaver is shown in Fig. 7.26. In addition to the concrete refinement of the IoT system part, the logical relationship between IoTApp-1 and External_API_1 on the application layer in the TCP/IP model is MQTT, where the IoTApp-1 is the source and External_API_1is the destination. The network layer protocol in the TCP/IP model is the IP, where the source is IoTNWIn (network interface WiFi) and the destination is External_API_1.

The WiFi network interface, IoT_Network_WiFi is connected to the L3SW,

```
(.venv) ubuntu@ip-172-16-20-23:~/work/git/weaver$ weaver -d ../weaver-data/ -r "rule_iot"
--intent-json backup/IoT-Test
Loading ['../weaver-data//types/component_type.json']
Loading ['../weaver-data//types/relationship_type.json']
Successfully loaded ../weaver-data//rules/rule_iot.json.
(0) action.db size: 16.0KiB  state.db size: 16.0KiB   0s
[]TERATION 20] 0.131306s elapsed
[]
No. of intent edge threats: 0
No. of intent node threats: 0

Skipped: <0> []

Design completed...

成功


Topology:
    Node:
        IoTApp-1: IoT_App
        IoT_HardwarePlatform<b>: IoT_ESP32
        IoT_NetworkInterface<c>: IoT_Network_WiFi
        IoT_OS<a>: IoT_FreeRTOS
        Switch<d>: L3SW
    Edge:
        IoTApp-1 -> IoT_OS<a>: wire:iot_os | {}
        IoT_HardwarePlatform<b> -> IoT_NetworkInterface<c>: wire:iot_network | {}
        IoT_NetworkInterface<c> -> Switch<d>: wire:lan | {}
        IoT_OS<a> -> IoT_HardwarePlatform<b>: wire:iot_hardware | {}

Total elapsed: 0.13796043395996094 s
Total security elapsed: 0.0008294582366943359 s
Total no. of threats: 0
Total no. of exit condition check: 21
Total no. of topology iteration: 32
Total no. of sec iteration: 1
Sec-topology iteration:
[(1, 32)]
shelve_path: ../weaver-data//actions.db  size: 264.0KiB
shelve_path: ../weaver-data//states.db  size: 768.0KiB
```

Figure 7.23: Output of standalone IoT system design by SecureWeaver.

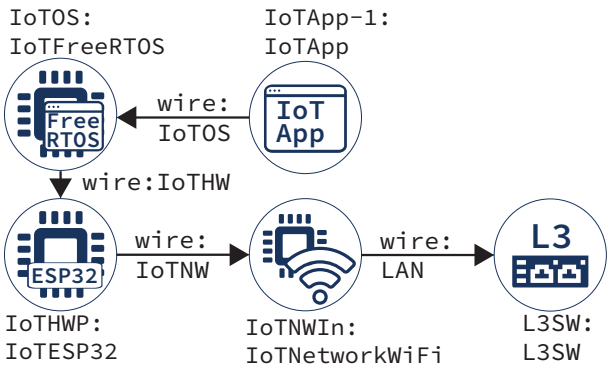

Figure 7.24: Output of standalone IoT system design by SecureWeaver.

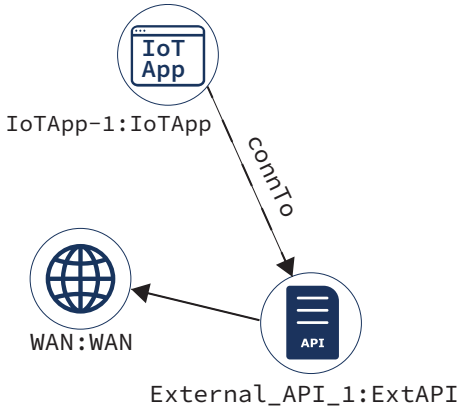

Figure 7.25: Service requirement of end-to-end IoT application system design.

where the L3SW is then connected to the Router via a concrete wire:lan type relationship. The WAN is connected to Router via a concrete wire:WAN-GW relationship, which completes the end-to-end path between the IoT application, IoTApp-1 and external API, External_API_1. This scenario highlights that SecureWeaver is not only able to design for IoT system as a standalone, but also its subsequent IT related networking part.

7.3.3.3 Evaluation of End-to-End IoT Application With One Security Threat

In this scenario, the end-to-end IoT application scenario service requirement is used to evaluate SecureWeaver's security verification mechanism. The MITRE ATT&CK Network Sniffing threat (T1040) is introduced into the end-to-end IoT application intent, where the abstract connTo relationship is explicitly defined to be vulnerable to the network sniffing threat as shown in Fig. 7.27. For the abstract connTo refinement to concrete MQTT connection, MQTT protocol by itself is not secure. Hence, SecureWeaver will reject the system design candidates that does not meet the required secure protocol to mitigate the threat T1040. The MQTT with TLS, MQTT_TLS is defined as a secure type of relationship in SecureWeaver database.

The result of the end-to-end IoT application scenario is shown in Fig. 7.28, where the topology edge section shows MQTT_TLS relationship type is defined for the relationship between the source, IoTApp-1 and destination, External_API_1. Note that the application layer protocol for the logical connection between IoTApp-1 and External_API_1 is MQTT_TLS, which is defined as one of the protocols that is suitable for the MITRE ATT&CK mitigation

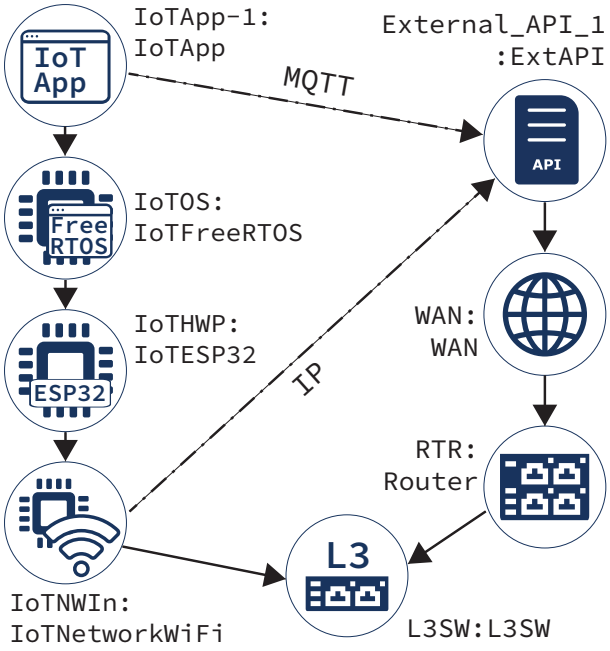

Figure 7.26: Output of end-to-end IoT application system design by SecureWeaver.

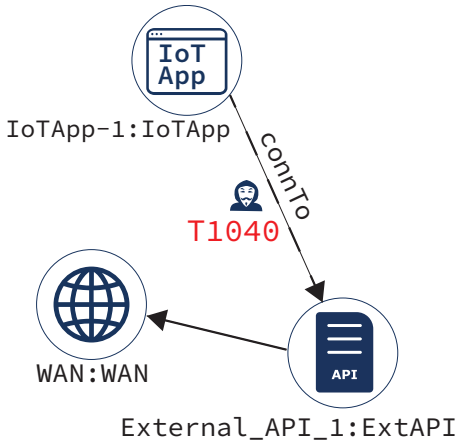

Figure 7.27: Service requirement of end-to-end IoT application system design with threat T1040.

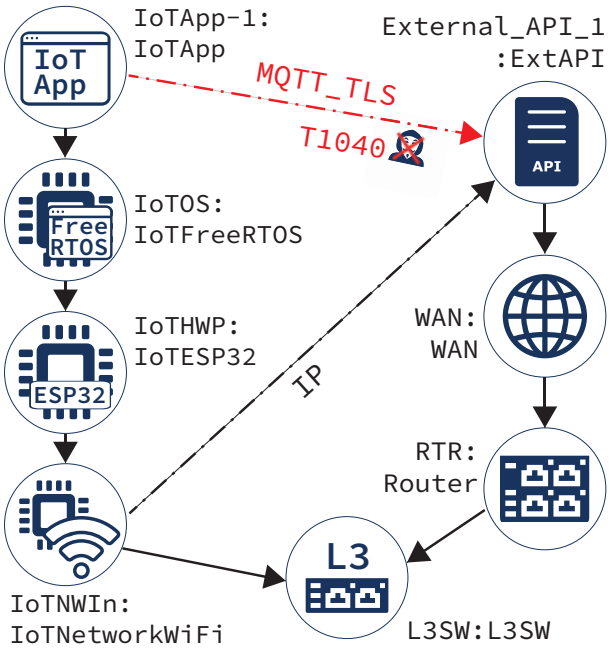

Figure 7.28: Output of end-to-end IoT application system design by SecureWeaver with security verification.

technique "Encrypt Sensitive Information" (M1041) in the SecureWeaver threat mitigation knowledge base. Since mitigation M1041 is mapped to threat T1040 (which appeared in the input requirement for this scenario), SecureWeaver utilizes the corresponding secure protocol verification function to check accordingly the system design candidates generated during the refinement process.

By analyzing the detailed SecureWeaver output, we were able to confirm that all system design candidates that did not meet the secure protocol requirement for mitigating threat T1040, such as `MQTT` at the application layer and `IP` at the network layer, were correctly rejected. Thus, this evaluation demonstrated that SecureWeaver is not only able to functionally design a concrete IoT system at both hardware and network levels, but can also ensure that the resulting IoT system design output is secure.

7.4 Feature Evaluation and Comparison

In this section, the feature evaluation of SecureWeaver and its comparison with related works are discussed. The discussion on the evaluation and the pros and cons of SecureWeaver are discussed in detail, which is followed by

the feature comparison between SecureWeaver and other system designers.

7.4.1 Evaluation of SecureWeaver Capabilities

The verification process in SecureWeaver was implemented according to the best practices included in the MITRE ATT&CK matrix and we have validated that these best practices are reflected in the output design. In addition, security professionals from NEC Corporation have assessed the SecureWeaver output design for several scenarios and they found it satisfactory from a security perspective. However, judging only the validity of the SecureWeaver output does not prove its effectiveness. Human designers often utilize checklists, such as the Functional Specification Document (FSD) and Technical Specification Document (TSD), to validate the functional and security requirements during design and implementation. In our method, instead of a checklist, the design is conducted based on security rules, and it is fundamentally equivalent to checklist-based design in term of its security characteristics. Moreover, SecureWeaver is not affected by the possible mistakes or omissions that human designers could make, and at the same time is able to verify significantly larger sets of conditions compared to human designers. On the other hand, a human designer can deal with various implicit requirements, whereas SecureWeaver can only handle the sets of explicit requirements that it is able to recognize (this aspect can be improved on by extending the set of refinement rules and the knowledge base).

In the current implementation of SecureWeaver, the output design is generated from a network domain perspective, which may not explicitly consider OS-dependent details, such as a certain security patches or vulnerabilities. Therefore, due diligence is required to ensure that the actual implementation of the system design is sound and follows the best security practices. To extend SecureWeaver beyond the network domain, security details at the OS level, such as security vulnerabilities and their mitigations can be included into the framework via additional refinement rules that include the relevant details. For instance, for network servers running Linux-based OSs, the Linux domain in the ATT&CK Enterprise matrix is relevant to ensure their security.

Besides that, it is possible for SecureWeaver to unknowingly introduce further vulnerabilities during its refinement process into the final system design output, since certain refinement rules may introduce a possible vector of attack in the system. Ideally, SecureWeaver should actively mitigate the threats introduced during the refinement process recursively. However due to the inherent limitation of SecureWeaver, it is only able to process and mitigate threats that are explicitly defined in the service requirement, and it

is unable to recursively mitigate new threats that are introduced as a result of the refinement.

The concept of "zero trust" has been gaining momentum in the security communities, and it means that components in a network are not trusted by default, and their interactions are always verified. Thus, the principle of least privilege recommends per-request access controls to achieve micro-segmentation, which requires enhanced identity management features. To apply the concept of zero trust to SecureWeaver would require a new meta-level verification function for access control regarding the relevant components, as well as additional refinement rules and an Access Control List (ACL) in the secure design database.

Like any system, SecureWeaver has its advantages and disadvantages, as summarized in Table 7.6. Regarding its advantages, SecureWeaver is able to automatically transform an abstract intent into a concrete system design that satisfies the input requirements, handling both the design-from-scratch and incremental-design cases. While SecureWeaver currently has built-in support for the IT/NW and IoT domains, it can also support any other type of networked system, as long as the corresponding components, relationships, and their refinement rules can be expressed in SecureWeaver format. SecureWeaver includes the MITRE ATT&CK based security verification functions that incorporate best practices for mitigating attacks against specific system components. Furthermore, SecureWeaver is not prone to mistakes and omissions as it can happen for human designers, and can verify significantly larger sets of conditions compared to a human designer.

As for disadvantages, SecureWeaver requires a technical user to define the service requirement, components, relationships and refinement rules for any system that is to be designed (if these elements are not already defined in its library). The current SecureWeaver only implements security verification and rules related to the network domain in MITRE ATT&CK, which limits the range of practical use scenarios. For example, threats such as Data Encrypted for Impact (T1486) that relate to ransomware attacks against database and backup servers are part of the Infrastructure-as-a-Service (IaaS), Linux, Windows, and macOS domains of MITRE ATT&CK; as they are outside the network domain currently implemented in SecureWeaver, such threats are not covered at this moment. In addition, SecureWeaver does not support automatic threat assignment or identification of cascading threat(s) from the included threats(s), as currently a security expert must explicitly define the security threat(s) of concern in the service requirement. Lastly, SecureWeaver can only handle a limited sets of requirements when compared to human designers, as the extent of the support depends on the amount of implemented refinement rules.

Table 7.6: Pros and cons of SecureWeaver.

Pros	Automatically transform an intent file into a concrete system design that satisfies the included qualitative, quantitative, and security requirements
	Can accept fully abstract or partially concrete intent files so that either full or incremental designs are possible
	Supports any type of networked system provided that the corresponding components, relationships, and their refinement rules can be expressed
	MITRE ATT&CK based security verification, which incorporates best practices for mitigating attacks against specific system components
	A smaller number of independent security verification functions and related rules compared to the number of threats, making their definition manageable
	Includes components, relationships and refinement rule definitions for the IT/NW and IoT domains
	Does not suffer common disadvantage of human designer and is able to verify significantly larger conditions
Cons	Requires the definitions of intent file, components, relationships and refinement rules for any system that is to be designed (if not already defined)
	Only the MITRE ATT&CK network domain security verification and related rules are currently implemented
	Requires the explicit inclusion of the security threat(s) of concern in the intent file; no automatic threat assignment or identification of cascading threat(s) from the included threat(s) is supported
	Only can handle limited sets of requirement when compared to human designers

7.4.2 Comparison with Related Works

In this section, SecureWeaver is compared to the related works that cover secure design aspects discussed in Section 2. The comparison first looks at each framework's capabilities, whether it is able to design or/and verify the security level of its output. The method that is used to design or create the output is also explored, as well as the type of output as its target domain. Furthermore, the method of how each framework creates or verifies its security policies/mitigations, whether qualitatively or quantitatively, and lastly, the type of security knowledge base that the framework refers to in order to refine its output are considered.

The comparison is summarized in Table 7.7. For framework capabilities, both SecureWeaver and [32] are able to perform both system design from abstract input and verify the security of the output, while the works in [30] and [31] are only capable of verifying a concretized input. As for [10], the framework is only capable of designing the security policy from an abstract input.

For the design method, both SecureWeaver and [32] use DSE refine a concrete system, while [30] uses clustering and external tools to manage its dependency process. The framework in [31] uses template matching to create the flow chain for verification, and [10] uses DFA and CFG to create concrete configurable security policies. Template-based approaches are generally more rigid than search-based design.

For the framework target domain, SecureWeaver covers both IT/NW and IoT aspects, as demonstrated in this paper and in [72]. However, the other frameworks target specific domains such as NW, IT/NW or IoT (the difference between NW and IT/NW is that NW only considers traffic routing in an SDN cloud environment, whereas IT/NW represents a more generic IT environment). Moreover, all the framework security threat mitigation approaches are quantitative-based except the framework in [10]. For the quantitative-based approach, a numerical result is typically used to satisfy the quantitative security requirements, while a qualitative-based approach such as [10] only creates its output with reference to a pattern database as a matching problem, which is not an optimization problem.

All platforms use some form of database which stores the ruleset/template/attack chain and their corresponding numerical values for security computations. While [30] and [31] are based on abstract numerical values, [32] designed their attack chain based on the STRIDE model. The SecureWeaver database is based on the ATT&CK matrix, which provides a more concrete and comprehensive coverage. The SecureWeaver database also allows users to assign numerical weights for each mitigation.

Table 7.7: Feature comparison of SecureWeaver with related works.

Name	Capability	Method	Target	Threat Mitigation	Security Knowledge
SecureWeaver	Design & verification	DSE	IT/NW, IoT	Quantitative (done via security verification & security level assessment)	ATT&CK database, security level & function database
INSpIRE [30]	Verification	Clustering	NW	Quantitative (done via security score computation)	Virtual Network Function (VNF) & security score database
[31]	Verification	Template	IT/NW	Quantitative (done via security level assessment)	Pattern & security level database
IBCS [10]	Design	DFA & CFG	NW	Qualitative (done via policy generation)	Network Security Function (NSF) database
[32]	Design & verification	DSE	IoT	Quantitative (done via probabilistic attack chain)	STRIDE based attack chain, security function database

This comparison demonstrates that SecureWeaver is well suited for addressing secure architecture-level system design in the IT/NW and IoT domains, and is favorably positioned compared to other related works.

7.5 Summary

This section presented the evaluation of SecureWeaver using two approaches: (i) functionality evaluation; and (ii) performance evaluation. A typical corporate network scenario that includes web systems, thin client systems, and remote user access was used to create multiple scenarios in increasing complexity and number of threats for the evaluation. SecureWeaver was shown to be able to generate a system design that mitigates the security threats included in the service requirement via the automatic assignment of a NIDS, a VPN server, and a firewall appliance at the suitable positions in the topology state, as well as utilizing a virtual machine environment to deploy a potentially-vulnerable web server application.

Furthermore, the performance characteristics of SecureWeaver implementation were also evaluated. The security verification overhead compared to the total elapsed time for system design is largest for simple scenarios, whereby the actual design process is very fast, still being just 0.58% in such a case. Nevertheless, the overhead for complex realistic scenarios with multiple threats sharply decreases, approaching levels as low as 0.30% in the experiments performed. A detailed discussion of the evaluation and the pros and cons of SecureWeaver is also discussed, followed by a feature comparison with respect to other research works, emphasising the overall advantages of SecureWeaver.

Chapter 8

Conclusion

8.1 Conclusion

The aim of this dissertation was to improve the security of a system by introducing formal verification of its security characteristics into fundamental system design, and also emphasizing practical experiment verification. The dissertation main contributions are summarized as follows:

1. The secure design database, as a part of the larger automated secure system designer framework, provides the database of secure refinement rules, MITRE ATT&CK-based threat mitigation knowledge base, and database of secure protocols. The secure design database provides the necessary information to the system designer, such that automated system designer is able to output a system design that is not only concrete, but also secure at the same time. An extension of the threat mitigation knowledge base was also presented, where using ontology and STIX, the threat mitigation knowledge base can be extended beyond MITRE ATT&CK to include other third-party security databases.
2. The intent-based secure system designer, SecureWeaver, realizes automated system design for networked systems that does not only meet the qualitative and quantitative requirements of the service requirement input, but also its security requirements. SecureWeaver was implemented by leveraging the functionality of an existing intent-based system designer that targeted IT/NW services, named Weaver, and introduced security verification mechanism into the verification process.
3. The case studies on secure end-to-end communication and secure configuration to demonstrate the development challenges with an IoT hardware platform. The end-to-end communication implementations covered both LPWAN Sigfox and MQTT to public cloud implementations, where the issues with Sigfox payload encryption and securing MQTT session were investigated and discussed. Furthermore, the work to port maker-friendly Arduino code into ESP-IDF in order to implement flash encryption was also presented. From the analysis, it

was shown that a flash encrypted ESP32 device may not be fully encrypted and discussed the pitfalls in ensuring the device is fully secure. These case studies demonstrated the challenges of implementing secure networked systems that guided the process of modelling threats and system components used for the evaluation of SecureWeaver.

4. A set of models for IT/NW and IoT system design is automatically designed via SecureWeaver to meet the qualitative, quantitative and security requirements. Practical scenarios such as corporate network, IoT appliance and hardware-level design are evaluated to showcase that it is possible to automatically design this kind of systems in a secure manner. In corporate network scenario, SecureWeaver was able to generate a system design that mitigates the security threats included in the input requirements via the automatic placement of a network intrusion detection system, a VPN server, and a firewall at the appropriate locations, as well as by deploying a potentially-vulnerable web server application in a virtual machine environment. The performance characteristics of SecureWeaver implementation demonstrated that the security verification overhead compared to the total system design time is largest for simple scenarios, for which the actual design is very fast, still being just 0.58% in such a case. However, for complex realistic scenarios with multiple threats the overhead decreases sharply, reaching levels as low as 0.30% in the experiments performed.

As of the social impact of this dissertation, the proposed automated secure system design is intended as a way to decrease the human effort required in system design via automatically designing secure systems. This would reduce the manual labour required to design secure system and redistribute the labour to higher value work. This research was conducted as a joint-research project with NEC Corporation to introducing security features into their existing system designer where several patent applications were made during this project, and the potential commercial use of the research by NEC will potentially increase its high impact in society.

8.2 Future Work

The following are possible future directions of this work:

- Since the current threat mitigation knowledge base is based only on the network domain in MITRE ATT&CK Enterprise matrix, the threat mitigation knowledge base could be extended to include other third-party security databases such as MITRE's Common Vulnerabilities and

Exposures (CVE), and many more.

- Artificial Intelligence (AI) can be used to optimize the automated system designer refinement process, making it faster and more efficient to generate larger and more complex system designs.
- Automated threat placement and rule generations can be introduced to reduce manual definition of threats in the service requirement and manual curation of security-focused refinement rules.
- Lastly, the automated system designer can be optimized regarding software performance by the early elimination of those system designs that cannot be made secure by any means.

References

[1] Industrial Internet Consortium (IIC). Trustworthiness in Industrial System Design. [Online]. Available: https://www.iiconsortium.org/news/joi-articles/2018-Sept-JoI-Trustworthiness-in-System-Design-Wibu-Systems.pdf

[2] C. Tucker, D. Shuffelton, R. Jhala, and S. Lerner, "Opium: Optimal package install/uninstall manager," in 29th International Conference on Software Engineering (ICSE'07). IEEE, 2007, pp. 178–188.

[3] F. Carrez, M. Bauer, M. Boussard, N. Bui, C. Jardak, J. Loof, C. Magerkurth, S. Meissner, A. Nettsträter, A. Olivereau et al., "Final architectural reference model for the iot v3. 0, internet of things-architecture iot-a ec project deliverable d1. 52013."

[4] OASIS Open. STIX 2.1 Examples. [Online]. Available: https://oasis-open.github.io/cti-documentation/stix/examples.html

[5] ——. STIX Version 2.1. [Online]. Available: https://docs.oasis-open.org/cti/stix/v2.1/csprd01/stix-v2.1-csprd01.html#_Toc16070624

[6] MITRE. ATT&CK STIX Data. [Online]. Available: https://github.com/mitre-attack/attack-stix-data/

[7] C. Wu, S. Horiuchi, K. Murase, H. Kikushima, and K. Tayama, "Intent-driven cloud resource design framework to meet cloud performance requirements and its application to a cloud-sensor system," Journal of Cloud Computing, vol. 10, no. 1, pp. 1–22, 2021.

[8] A. Rafiq, A. Mehmood, T. Ahmed Khan, K. Abbas, M. Afaq, and W.-C. Song, "Intent-based end-to-end network service orchestration system for multi-platforms," Sustainability, vol. 12, no. 7, p. 2782, 2020.

[9] M. Pham and D. B. Hoang, "Sdn applications-the intent-based northbound interface realisation for extended applications," in 2016 IEEE NetSoft Conference and Workshops (NetSoft). IEEE, 2016, pp. 372–377.

[10] J. Kim, E. Kim, J. Yang, J. Jeong, H. Kim, S. Hyun, H. Yang, J. Oh, Y. Kim, S. Hares et al., "Ibcs: intent-based cloud services for security applications," IEEE Communications Magazine, vol. 58, no. 4, pp. 45–51, 2020.

[11] A. S. Jacobs, R. J. Pfitscher, R. H. Ribeiro, R. A. Ferreira, L. Z. Granville, W. Willinger, and S. G. Rao, "Hey, lumi! using natural language for {Intent-Based} network management," in 2021 USENIX Annual Technical Conference (USENIX ATC 21), 2021, pp. 625–639.

[12] C. El Houssaini, M. Nassar, and A. Kriouile, "A cloud service template for enabling accurate cloud adoption and migration," in 2015 International Conference on Cloud Technologies and Applications (CloudTech). IEEE, 2015, pp. 1–6.

[13] J. DesLauriers, T. Kiss, G. Pierantoni, G. Gesmier, and G. Terstyanszky, "Enabling modular design of an application-level auto-scaling and orchestration framework using TOSCA-based application description templates," in 11th International Workshop on Science Gateways, IWSG 2019. CEUR Workshop Proceedings, 2021.

[14] N. Paladi, A. Michalas, and H.-V. Dang, "Towards secure cloud orchestration for multi-cloud deployments," in Proceedings of the 5th Workshop on CrossCloud Infrastructures & Platforms, 2018, pp. 1–6.

[15] Y. Wei, M. Peng, and Y. Liu, "Intent-based networks for 6g: Insights and challenges," Digital Communications and Networks, vol. 6, no. 3, pp. 270–280, 2020.

[16] E. Kang, "Design space exploration for security," in 2016 IEEE Cybersecurity Development (SecDev). IEEE, 2016, pp. 30–36.

[17] A. D. Pimentel, "A case for security-aware design-space exploration of embedded systems," Journal of Low Power Electronics and Applications, vol. 10, no. 3, p. 22, 2020.

[18] T. Kuroda, T. Kuwahara, T. Maruyama, K. Satoda, H. Shimonishi, T. Osaki, and K. Matsuda, "Weaver: A novel configuration designer for it/nw services in heterogeneous environments," in 2019 IEEE Global Communications Conference (GLOBECOM). IEEE, 2019, pp. 1–6.

[19] T. Kuwahara, T. Kuroda, T. Osaki, and K. Satoda, "An intent-based system configuration design for it/nw services with functional and

quantitative constraints," IEICE Transactions on Communications, vol. E104.B, no. 7, pp. 791–804, 2021.

[20] J. Luszcz, "Apache struts 2: how technical and development gaps caused the equifax breach," Network Security, vol. 2018, no. 1, pp. 5–8, 2018.

[21] G. McGraw, "Software security," Building security in, 2006.

[22] J. C. Santos, K. Tarrit, and M. Mirakhorli, "A catalog of security architecture weaknesses," in 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). IEEE, 2017, pp. 220–223.

[23] T. Pollock, "Reducing human error in cyber security using the human factors analysis classification system (hfacs)." 2017.

[24] R. Kwon, T. Ashley, J. Castleberry, P. Mckenzie, and S. N. G. Gourisetti, "Cyber threat dictionary using mitre att&ck matrix and nist cybersecurity framework mapping," in 2020 Resilience Week (RWS). IEEE, 2020, pp. 106–112.

[25] A. Georgiadou, S. Mouzakitis, and D. Askounis, "Assessing mitre att&ck risk using a cyber-security culture framework," Sensors, vol. 21, no. 9, p. 3267, 2021.

[26] A. Brazhuk, "Semantic model of attacks and vulnerabilities based on capec and cwe dictionaries," International Journal of Open Information Technologies, vol. 7, no. 3, pp. 38–41, 2019.

[27] W. Xiong, E. Legrand, O. Åberg, and R. Lagerström, "Cyber security threat modeling based on the mitre enterprise att&ck matrix," Software and Systems Modeling, vol. 21, no. 1, pp. 157–177, 2022.

[28] E. Hemberg, J. Kelly, M. Shlapentokh-Rothman, B. Reinstadler, K. Xu, N. Rutar, and U.-M. O'Reilly, "Bron–linking attack tactics, techniques, and patterns with defensive weaknesses, vulnerabilities and affected platform configurations," arXiv e-prints, pp. arXiv–2010, 2020.

[29] G. Davoli, W. Cerroni, S. Tomovic, C. Buratti, C. Contoli, and F. Callegati, "Intent-based service management for heterogeneous software-defined infrastructure domains," International Journal of Network Management, vol. 29, no. 1, p. e2051, 2019.

[30] E. J. Scheid, C. C. Machado, M. F. Franco, R. L. dos Santos, R. P. Pfitscher, A. E. Schaeffer-Filho, and L. Z. Granville, "Inspire: Integrated nfv-based intent refinement environment," in 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). IEEE, 2017, pp. 186–194.

[31] F. Amato, N. Mazzocca, and F. Moscato, "Model driven design and evaluation of security level in orchestrated cloud services," Journal of Network and Computer Applications, vol. 106, pp. 78–89, 2018.

[32] L. Gressl, C. Steger, and U. Neffe, "Design space exploration for secure iot devices and cyber-physical systems," ACM Transactions on Embedded Computing Systems (TECS), vol. 20, no. 4, pp. 1–24, 2021.

[33] M. Rutkowski, C. Chris Lauwers, and C. Curescu, "Tosca simple profile in yaml version 1.3," 2020. [Online]. Available: https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/TOSCA-Simple-Profile-YAML-v1.3.pdf

[34] D. Makoshenko and I. Enkovich, "Iot development: Discovering, enabling and validation of real life iot scenarios," in 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC). IEEE, 2017, pp. 159–164.

[35] A. R. DeSerranno, M. T. Mullarkey, and A. R. Hevner, "Building a semantic ontology for internet of things (iot) systems." in ONTOBRAS, 2017, pp. 111–117.

[36] D. Amalfitano, N. Amatucci, V. De Simone, V. Riccio, and F. A. Rita, "Towards a thing-in-the-loop approach for the verification and validation of iot systems," in Proceedings of the 1st ACM Workshop on the Internet of Safe Things, 2017, pp. 57–63.

[37] H. Kim, A. Ahmad, J. Hwang, H. Baqa, F. Le Gall, M. A. R. Ortega, and J. Song, "Iot-taas: Towards a prospective iot testing framework," IEEE Access, vol. 6, pp. 15 480–15 493, 2018.

[38] X. Mountrouidou, B. Billings, and L. Mejia-Ricart, "Not just another internet of things taxonomy: A method for validation of taxonomies," Internet of Things, vol. 6, p. 100049, 2019.

[39] J. Voas et al., "Networks of 'things'," NIST Special Publication, vol. 800, no. 183, pp. 800–183, 2016.

[40] L. Maillet-Contoz, E. Michel, M. D. Nava, P.-E. Brun, K. Leprêtre, and G. Massot, "End-to-end security validation of iot systems based on digital twins of end-devices," in 2020 Global Internet of Things Summit (GIoTS). IEEE, 2020, pp. 1–6.

[41] L. Gressl, M. Krisper, C. Steger, and U. Neffe, "Towards an automated exploration of secure iot/cps design-variants," in International Conference on Computer Safety, Reliability, and Security. Springer, 2020, pp. 372–386.

[42] C. Alberca, S. Pastrana, G. Suarez-Tangil, and P. Palmieri, "Security analysis and exploitation of arduino devices in the internet of things," in Proceedings of the ACM International Conference on Computing Frontiers, 2016, pp. 437–442.

[43] A. A. Gendreau, "Internet of things: Arduino vulnerability analysis," A Primer for Security, p. 32, 2016.

[44] T. Niemirepo, M. Sihvonen, V. Jordan, and J. Heinilä, "Service platform for automated iot service provisioning," in 2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. IEEE, 2015, pp. 325–329.

[45] E. Yigitoglu, M. Mohamed, L. Liu, and H. Ludwig, "Foggy: a framework for continuous automated iot application deployment in fog computing," in 2017 IEEE international conference on AI & Mobile Services (AIMS). IEEE, 2017, pp. 38–45.

[46] A. R. DeSerranno, M. Mullarkey, and A. Hevner, "Evaluation of a commercial iot platform," 2017.

[47] Bosch. Bosch IoT Suite. [Online]. Available: https://www.bosch-iot-suite.com

[48] IBM. IBM Watson IoT Platform. [Online]. Available: https://internetofthings.ibmcloud.com/

[49] Microtronics. Microtronics IoT Suite. [Online]. Available: https://www.microtronics.com/en/product/iot-suite.html

[50] D. Soukaras, P. Patel, H. Song, and S. Chaudhary, "Iotsuite: a toolsuite for prototyping internet of things applications," in The 4th International Workshop on Computing and Networking for Internet of Things (ComNet-IoT), co-located with 16th International Conference on Distributed Computing and Networking (ICDCN), 2015, p. 6.

[51] G. M. Køien, "A philosophy of security architecture design," Wireless Personal Communications, vol. 113, no. 3, pp. 1615–1639, 2020.

[52] Industrial Internet Consortium (IIC). The Industrial Internet of Things: Managing and Assessing Trustworthiness for IIoT in Practice. [Online]. Available: https://www.iiconsortium.org/pdf/Managing_and_Assessing_Trustworthiness_for_IIoT_in_Practice_Whitepaper_20190516.pdf

[53] V. Shannon and W. J. Rewak. An Introduction to Cybersecurity Ethics. [Online]. Available: https://www.scu.edu/media/ethics-center/technology-ethics/IntroToCybersecurityEthics.pdf

[54] R. Beuran, S. E. Ooi, A. Barbir, and Y. Tan, "POSTER: IoT System Trustworthiness Assurance," in Posters In 2022 ACM Asia Conference on Computer and Communications Security (AsiaCCS), 2022.

[55] International Organization for Standardization/International Electrotechnical Commission (ISO/IEC). ISO/IEC JTC 1/SC 41: Internet of things and digital twin. [Online]. Available: https://www.iso.org/committee/6483279.html

[56] National Institute of Standards and Technology (NIST). NIST Cybersecurity for IoT Program web page. [Online]. Available: https://www.nist.gov/itl/applied-cybersecurity/nist-cybersecurity-iot-program

[57] Industrial Internet Consortium (IIC). Industrial Internet Consortium webpage. [Online]. Available: https://www.iiconsortium.org

[58] E. R. Griffor, C. Greer, D. A. Wollman, M. J. Burns et al., "Framework for cyber-physical systems: Volume 1, overview," 2017.

[59] A. Karmarkar and M. Buchheit, "The industrial internet of things volume g8: Vocabulary," IIC: PUB G, vol. 8, 2017.

[60] International Organization for Standardization, "Road vehicles - Functional safety. ISO 26262:2018," 2018.

[61] Japan Automobile Manufacturers Association (JAMA). Self-driving safety assessment framework (in Japanese). [Online]. Available: https://www.jama.or.jp/safe/automated_driving/pdf/framework.pdf

[62] European Union Agency for Cybersecurity (ENISA). Good Practices for Security of IoT – Secure Software Development Lifecycle. [Online]. Available: https://www.enisa.europa.eu/publications/good-practices-for-security-of-iot-1/@@download/fullReport

[63] National Institute of Standards and Technology (NIST). IoT Device Cybersecurity Requirement Catalogs. [Online]. Available: https://pages.nist.gov/IoT-Device-Cybersecurity-Requirement-Catalogs/

[64] U.S. Department of Health & Human Services. Health Insurance Portability and Accountability Act (HIPAA). [Online]. Available: https://www.hhs.gov/hipaa/index.html

[65] European Parliament and Council of the European Union. General Data Protection Regulation (GDPR) 2016/679. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj

[66] M. Rausand, A. Barros, and A. Hoyland, System reliability theory: models, statistical methods, and applications (3rd edition). John Wiley & Sons, 2020.

[67] P. Uday and K. Marais, "Designing resilient systems-of-systems: A survey of metrics, methods, and challenges," Systems Engineering, vol. 18, no. 5, pp. 491–510, 2015.

[68] G. Marshall and D. Chapman. Resilience, Reliability and Redundancy. [Online]. Available: http://copperalliance.org.uk/uploads/2018/03/41-resilience-reliability-and-redundancy.pdf

[69] A. Taivalsaari and T. Mikkonen, "A taxonomy of iot client architectures," IEEE software, vol. 35, no. 3, pp. 83–88, 2018.

[70] V. C. Pham, Y. Lim, A. Sgorbissa, and Y. Tan, "An ontology-driven echonet lite adaptation layer for smart homes," Journal of Information Processing, vol. 27, pp. 360–368, 2019.

[71] European Telecommunications Standards Institute (ETSI). Smart Appliances Reference Ontology and oneM2M Mapping. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/103200_103299/103264/02

[72] S. E. Ooi, R. Beuran, Y. Tan, T. Kuroda, T. Kuwahara, and N. Fujita, "Secureweaver: Intent-driven secure system designer," in Proceedings of the 2022 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems, 2022, pp. 107–116.

[73] S. E. Ooi, R. Beuran, T. Kuroda, T. Kuwahara, R. Hotchi, N. Fujita, and Y. Tan, "Intent-driven secure system design: Methodology and implementation," Computers & Security, vol. 124, p. 102955, 2023.

[74] Lockheed Martin, "Cyber kill chain," 2014. [Online]. Available: http://cyber.lockheedmartin.com/hubfs/GainingtheAdvantageCyberKillChain.pdf

[75] B. Strom, A. Applebaum, D. Miller, K. Nickels, A. Pennington, and C. Thomas, "Mitre att&ck: Design and philosophy. the mitre corporation, mclean," VA, Technical report, Tech. Rep., 2018.

[76] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, "Uncover security design flaws using the stride approach." Microsoft Corporation, 2006.

[77] S. Barnum, "Standardizing cyber threat intelligence information with the structured threat information expression (stix)," Mitre Corporation, vol. 11, pp. 1–22, 2012.

[78] MITRE. MITRE ATT&CK. [Online]. Available: https://attack.mitre.org

[79] ——. MITRE Common Attack Pattern Enumeration and Classification. [Online]. Available: http://capec.mitre.org

[80] ——. MITRE Common Vulnerabilities and Exposure. [Online]. Available: https://cve.mitre.org

[81] ——. MITRE Common Weakness Enumeration. [Online]. Available: https://attack.mitre.org

[82] National Institute of Standards and Technology (NIST). NIST National Vulnerability Database (NVD). [Online]. Available: https://nvd.nist.gov

[83] R. Stillions, "The dml model," Ryan Stillions: Postulations after great cogitation, 2014.

[84] V. Mavroeidis and S. Bromander, "Cyber threat intelligence model: an evaluation of taxonomies, sharing standards, and ontologies within cyber threat intelligence," in 2017 European Intelligence and Security Informatics Conference (EISIC). IEEE, 2017, pp. 91–98.

[85] W. Gibb and D. Kerr, "Openioc: back to the basics," OpenIOC: Back to the Basics, 2018.

[86] R. Danyliw, J. Meijer, Y. Demchenko et al., "The incident object description exchange format," IETF Request For Comments, vol. 5070, 2007.

[87] Verizon Security Research & Cyber Intelligence Center. Vocabulary for Event Recording and Incident Sharing (VERIS). [Online]. Available: https://github.com/vz-risk/veris

[88] F. Böhm, F. Menges, and G. Pernul, "Graph-based visual analytics for cyber threat intelligence," Cybersecurity, vol. 1, no. 1, pp. 1–19, 2018.

[89] C. Sauerwein, C. Sillaber, A. Mussmann, and R. Breu, "Threat intelligence sharing platforms: An exploratory study of software vendors and research perspectives," 2017.

[90] Mandiant. Poison Ivy: Assessing Damage and Extracting Intelligence. [Online]. Available: https://www.mandiant.com/resources/poison-ivy-assessing-damage-and-extracting-intelligence

[91] MITRE. CTI. [Online]. Available: https://github.com/mitre/cti

[92] S. E. Ooi. Codes related to Automated Secure Design of Networked Systems. [Online]. Available: https://github.com/quadcube/Codes-for-Automated-Secure-Design-for-Networked-Systems

[93] S. E. Ooi, R. Beuran, and Y. Tan, "Secure iot development: A maker's perspective," in 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS). IEEE, 2021, pp. 1–6.

[94] IBM. Choosing the best hardware for your next IoT project. [Online]. Available: https://developer.ibm.com/articles/iot-lp101-best-hardware-devices-iot-project/

[95] IoT Rapid-Proto Labs. WP3: Internet of Things: Best Practices in Technology, Development Methods and Product Design. [Online]. Available: https://www.rapidprotolabs.eu/wp-content/uploads/2019/02/WP3-Report-Best-Practices-Report.pdf

[96] W. M. Stout and V. E. Urias, "Challenges to securing the internet of things," in 2016 IEEE International Carnahan Conference on Security Technology (ICCST). IEEE, 2016, pp. 1–8.

[97] R. Fujdiak, P. Blazek, K. Mikhaylov, L. Malina, P. Mlynek, J. Misurec, and V. Blazek, "On track of sigfox confidentiality with end-to-end

encryption," in Proceedings of the 13th International Conference on Availability, Reliability and Security, 2018, pp. 1–6.

[98] Sigfox, "Sigfox technical overview," 2017.

[99] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, "Long-range communications in unlicensed bands: The rising stars in the iot and smart city scenarios," IEEE Wireless Communications, vol. 23, no. 5, pp. 60–67, 2016.

[100] M. Calabretta, R. Pecori, M. Vecchio, and L. Veltri, "Mqtt-auth: A token-based solution to endow mqtt with authentication and authorization capabilities," Journal of Communications Software and Systems, vol. 14, no. 4, pp. 320–331, 2018.

[101] Espressif. To use as a component of ESP-IDF. [Online]. Available: https://github.com/espressif/arduino-esp32/blob/master/docs/esp-idf_component.md

Publications

[1] Sian En Ooi, Razvan Beuran, Takayuki Kuroda, Takuya Kuwahara, Ryosuke Hotchi, Norihito Fujita, and Yasuo Tan. 2023. Automated Secure Design with Machine Learning. [Planning to submit to ESORICS 2023]

[2] Sian En Ooi, Razvan Beuran, Takayuki Kuroda, Takuya Kuwahara, Norihito Fujita, and Yasuo Tan. 2023. Study on Automated Secure Design for IoT Systems. [Planning to submit to SafeComp 2023]

[3] Sian En Ooi, Razvan Beuran, Takayuki Kuroda, Takuya Kuwahara, Ryosuke Hotchi, Norihito Fujita, and Yasuo Tan. 2023. Intent-Driven Secure System Design: Methodology and Implementation. In Computers & Security, 124, 102955, 1st January 2023.

[4] Razvan Beuran, Sian En Ooi, Abbie Barbir, and Yasuo Tan. 2022. POSTER: IoT System Trustworthiness Assurance. In ACM Asia Conference on Computer and Communications Security (AsiaCCS) 2022 Poster Session, 30th May 2022.

[5] Sian En Ooi, Razvan Beuran, Yasuo Tan, Takayuki Kuroda, Takuya Kuwahara, and Norihito Fujita. 2022. SecureWeaver: Intent-Driven Secure System Configuration Designer. In Proceedings of the 2022 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems. April 2022.

[6] Sian En Ooi, Razvan Beuran, and Yasuo Tan. 2021. Secure IoT Development – A Maker's Perspective. In 2021 IEEE COINS, pp. 1-6. 23th August 2021.

[7] Yuan Fang, Yuto Lim, Sian En Ooi, Chen Zhou, and Yasuo Tan. 2020. Study of human thermal comfort for cyber-physical human centric system in smart homes. In 2020 Sensors, 20(2), pp. 372. January 2020.

[8] Sian En Ooi, Yuan Fang, Yuto Lim, and Yasuo Tan. 2019. Study of adaptive model predictive control for cyber-physical systems. In 2019 ICCST, pp. 165 – 174. 29th August 2018.

[9] Yuan Fang, Sian En Ooi, Yuto Lim, and Yasuo Tan. 2019. Time task scheduling for simple and proximate time model in cyber-physical systems. In 2019 ICCST, pp. 185 – 194. 29th August 2018.

[10] Sian En Ooi, Yoshiki Makino, Yuto Lim, and Yasuo Tan. 2018. Predictive thermal comfort control for cyber-physical smart home systems. In 2018 IEEE SoSE, pp. 444 – 451. 19th June 2018.

[11] Sian En Ooi, Yamin Thiri Aung, Yuan Fang, Yuto Lim, and Yasuo Tan. 2019. Implementation of predictive thermal comfort control for cyber-physical home systems. In IEICE technical report 118 (468), pp. 131 – 136. 5th March 2019.

[12] Sian En Ooi, Yoshiki Makino, Yuan Fang, Yuto Lim, and Yasuo Tan. 2018. Study of predictive thermal comfort control for cyber-physical smart home systems. In IEICE technical report 117 (426), pp. 29 – 34. 30th January 2018.

Appendix A

IoT Refinement Rules

```
{
    "rules": [
        {
            "rid": "LAN to Switch",
            "target": {
                "nid": "nid_LAN_1",
                "typ": "LAN"
            },
            "lhs": [
                {
                    "nid": "nid_LAN_1",
                    "typ": "LAN"
                }
            ],
            "rhs": [
                {
                    "nid": "nid_LAN_1",
                    "typ": "Switch"
                }
            ]
        },
        {
            "rid": "Switch to L3SW",
            "target": {
                "nid": "nid_Switch_1",
                "typ": "Switch"
            },
            "lhs": [
                {
                    "nid": "nid_Switch_1",
                    "typ": "Switch"
                }
            ],
            "rhs": [
                {
                    "nid": "nid_Switch_1",
                    "typ": "L3SW"
                }
```

```
            ]
        },
        {
            "rid": "Switch to L2SW",
            "target": {
                "nid": "nid_Switch_1",
                "typ": "Switch"
            },
            "lhs": [
                {
                    "nid": "nid_Switch_1",
                    "typ": "Switch"
                }
            ],
            "rhs": [
                {
                    "nid": "nid_Switch_1",
                    "typ": "L2SW"
                }
            ]
        },
        {
            "rid": "[JAIST] IoT App to IoT OS",
            "target": {
                "nid": "nid_IoTApp_1",
                "typ": "IoT_App"
            },
            "lhs": [
                {
                    "nid": "nid_IoTApp_1",
                    "typ": "IoT_App"
                }
            ],
            "rhs": [
                {
                    "nid": "nid_IoTApp_1",
                    "typ": "IoT_App"
                },
                {
                    "nid": "nid_IoTOS_1",
                    "typ": "IoT_OS"
                },
                {
                    "typ": "wire:iot_os",
                    "src": "nid_IoTApp_1",
                    "dst": "nid_IoTOS_1"
                }
            ]
        },
```

```
88          {
89              "rid": "[JAIST] IoT OS to IoT Hardware Platform",
90              "target": {
91                  "nid": "nid_IoTOS_1",
92                  "typ": "IoT_OS"
93              },
94              "lhs": [
95                  {
96                      "nid": "nid_IoTOS_1",
97                      "typ": "IoT_OS"
98                  }
99              ],
100             "rhs": [
101                 {
102                     "nid": "nid_IoTOS_1",
103                     "typ": "IoT_OS"
104                 },
105                 {
106                     "nid": "nid_IoTHardwarePlatform_1",
107                     "typ": "IoT_HardwarePlatform"
108                 },
109                 {
110                     "typ": "wire:iot_hardware",
111                     "src": "nid_IoTOS_1",
112                     "dst": "nid_IoTHardwarePlatform_1"
113                 }
114             ]
115         },
116         {
117             "rid": "[JAIST] IoT Hardware Platform to IoT Network
                    Interface",
118             "target": {
119                 "nid": "nid_IoTHardwarePlatform_1",
120                 "typ": "IoT_HardwarePlatform"
121             },
122             "lhs": [
123                 {
124                     "nid": "nid_IoTHardwarePlatform_1",
125                     "typ": "IoT_HardwarePlatform"
126                 }
127             ],
128             "rhs": [
129                 {
130                     "nid": "nid_IoTHardwarePlatform_1",
131                     "typ": "IoT_HardwarePlatform"
132                 },
133                 {
134                     "nid": "nid_IoTNetworkInterface_1",
135                     "typ": "IoT_NetworkInterface"
```

```
                },
                {
                    "typ": "wire:iot_network",
                    "src": "nid_IoTHardwarePlatform_1",
                    "dst": "nid_IoTNetworkInterface_1"
                }
            ]
        },
        {
            "rid": "[JAIST] IoT OS to FreeRTOS",
            "target": {
                "nid": "nid_OS_1",
                "typ": "OS"
            },
            "lhs": [
                {
                    "nid": "nid_IoTOS_1",
                    "typ": "IoT_OS"
                }
            ],
            "rhs": [
                {
                    "nid": "nid_IoTOS_1",
                    "typ": "IoT_FreeRTOS"
                }
            ]
        },
        {
            "rid": "[JAIST] IoT Hardware Platform to ESP32",
            "target": {
                "nid": "nid_IoTHardwarePlatform_1",
                "typ": "IoT_HardwarePlatform"
            },
            "lhs": [
                {
                    "nid": "nid_IoTHardwarePlatform_1",
                    "typ": "IoT_HardwarePlatform"
                }
            ],
            "rhs": [
                {
                    "nid": "nid_IoTHardwarePlatform_1",
                    "typ": "IoT_ESP32"
                }
            ]
        },
        {
            "rid": "[JAIST] IoT Network Interface to WiFi",
            "target": {
```

```
185                 "nid": "nid_IoTNetworkInterface_1",
186                 "typ": "IoT_NetworkInterface"
187             },
188             "lhs": [
189                 {
190                     "nid": "nid_IoTNetworkInterface_1",
191                     "typ": "IoT_NetworkInterface"
192                 }
193             ],
194             "rhs": [
195                 {
196                     "nid": "nid_IoTNetworkInterface_1",
197                     "typ": "IoT_Network_WiFi"
198                 }
199             ]
200         },
201         {
202             "rid": "[JAIST] IoT Network Interface to Switch",
203             "target": {
204                 "nid": "nid_IoTNetworkInterface_1",
205                 "typ": "IoT_NetworkInterface"
206             },
207             "lhs": [
208                 {
209                     "nid": "nid_IoTNetworkInterface_1",
210                     "typ": "IoT_NetworkInterface"
211                 }
212             ],
213             "rhs": [
214                 {
215                     "nid": "nid_IoTNetworkInterface_1",
216                     "typ": "IoT_NetworkInterface"
217                 },
218                 {
219                     "nid": "nid_Switch_1",
220                     "typ": "Switch"
221                 },
222                 {
223                     "typ": "wire:lan",
224                     "src": "nid_IoTNetworkInterface_1",
225                     "dst": "nid_Switch_1"
226                 }
227             ]
228         },
229         {
230             "rid": "[JAIST] IoT App to ExternalAPI (MQTT only)",
231             "target": {
232                 "typ": "connTo",
233                 "src": "nid_IoTApp_1",
```

```
234                     "dst": "nid_ExternalThings_1"
235                 },
236                 "lhs": [
237                     {
238                         "nid": "nid_ExternalThings_1",
239                         "typ": "ExternalThings"
240                     },
241                     {
242                         "typ": "connTo",
243                         "src": "nid_IoTApp_1",
244                         "dst": "nid_ExternalThings_1"
245                     },
246                     {
247                         "nid": "nid_IoTApp_1",
248                         "typ": "IoT_App"
249                     },
250                     {
251                         "typ": "wire:iot_os",
252                         "src": "nid_IoTApp_1",
253                         "dst": "nid_IoTOS_1"
254                     },
255                     {
256                         "nid": "nid_IoTOS_1",
257                         "typ": "IoT_OS"
258                     },
259                     {
260                         "typ": "wire:iot_hardware",
261                         "src": "nid_IoTOS_1",
262                         "dst": "nid_IoTHardwarePlatform_1"
263                     },
264                     {
265                         "nid": "nid_IoTHardwarePlatform_1",
266                         "typ": "IoT_HardwarePlatform"
267                     },
268                     {
269                         "typ": "wire:iot_network",
270                         "src": "nid_IoTHardwarePlatform_1",
271                         "dst": "nid_IoTNetworkInterface_1"
272                     },
273                     {
274                         "nid": "nid_IoTNetworkInterface_1",
275                         "typ": "IoT_NetworkInterface"
276                     },
277                     {
278                         "typ": "wire:lan",
279                         "src": "nid_IoTNetworkInterface_1",
280                         "dst": "nid_Switch_1"
281                     },
282                     {
```

```
283                     "nid": "nid_Switch_1",
284                     "typ": "Switch"
285                 },
286                 {
287                     "nid": "nid_WAN_1",
288                     "typ": "WAN"
289                 },
290                 {
291                     "typ": "wire:external-GW",
292                     "src": "nid_ExternalThings_1",
293                     "dst": "nid_WAN_1"
294                 }
295             ],
296             "rhs": [
297                 {
298                     "nid": "nid_IoTApp_1",
299                     "typ": "IoT_App"
300                 },
301                 {
302                     "typ": "wire:iot_os",
303                     "src": "nid_IoTApp_1",
304                     "dst": "nid_IoTOS_1"
305                 },
306                 {
307                     "nid": "nid_IoTOS_1",
308                     "typ": "IoT_OS"
309                 },
310                 {
311                     "typ": "wire:iot_hardware",
312                     "src": "nid_IoTOS_1",
313                     "dst": "nid_IoTHardwarePlatform_1"
314                 },
315                 {
316                     "nid": "nid_IoTHardwarePlatform_1",
317                     "typ": "IoT_HardwarePlatform"
318                 },
319                 {
320                     "typ": "wire:iot_network",
321                     "src": "nid_IoTHardwarePlatform_1",
322                     "dst": "nid_IoTNetworkInterface_1"
323                 },
324                 {
325                     "nid": "nid_IoTNetworkInterface_1",
326                     "typ": "IoT_NetworkInterface"
327                 },
328                 {
329                     "typ": "wire:lan",
330                     "src": "nid_IoTNetworkInterface_1",
331                     "dst": "nid_Switch_1"
```

```
332                 },
333                 {
334                     "nid": "nid_Switch_1",
335                     "typ": "Switch"
336                 },
337                 {
338                     "typ": "wire:lan",
339                     "src": "nid_Router_1",
340                     "dst": "nid_Switch_1"
341                 },
342                 {
343                     "nid": "nid_Router_1",
344                     "typ": "Router"
345                 },
346                 {
347                     "typ": "wire:WAN-GW",
348                     "src": "nid_WAN_1",
349                     "dst": "nid_Router_1"
350                 },
351                 {
352                     "nid": "nid_WAN_1",
353                     "typ": "WAN"
354                 },
355                 {
356                     "typ": "wire:external-GW",
357                     "src": "nid_ExternalThings_1",
358                     "dst": "nid_WAN_1"
359                 },
360                 {
361                     "nid": "nid_ExternalThings_1",
362                     "typ": "ExternalThings"
363                 },
364                 {
365                     "typ": "MQTT",
366                     "src": "nid_IoTApp_1",
367                     "dst": "nid_ExternalThings_1"
368                 },
369                 {
370                     "typ": "IP",
371                     "src": "nid_IoTNetworkInterface_1",
372                     "dst": "nid_ExternalThings_1"
373                 }
374             ]
375         },
376         {
377             "rid": "[JAIST] IoT App to ExternalAPI (MQTT-TLS)",
378             "target": {
379                 "typ": "connTo",
380                 "src": "nid_IoTApp_1",
```

```
                "dst": "nid_ExternalThings_1"
            },
            "lhs": [
                {
                    "nid": "nid_ExternalThings_1",
                    "typ": "ExternalThings"
                },
                {
                    "typ": "connTo",
                    "src": "nid_IoTApp_1",
                    "dst": "nid_ExternalThings_1"
                },
                {
                    "nid": "nid_IoTApp_1",
                    "typ": "IoT_App"
                },
                {
                    "typ": "wire:iot_os",
                    "src": "nid_IoTApp_1",
                    "dst": "nid_IoTOS_1"
                },
                {
                    "nid": "nid_IoTOS_1",
                    "typ": "IoT_OS"
                },
                {
                    "typ": "wire:iot_hardware",
                    "src": "nid_IoTOS_1",
                    "dst": "nid_IoTHardwarePlatform_1"
                },
                {
                    "nid": "nid_IoTHardwarePlatform_1",
                    "typ": "IoT_HardwarePlatform"
                },
                {
                    "typ": "wire:iot_network",
                    "src": "nid_IoTHardwarePlatform_1",
                    "dst": "nid_IoTNetworkInterface_1"
                },
                {
                    "nid": "nid_IoTNetworkInterface_1",
                    "typ": "IoT_NetworkInterface"
                },
                {
                    "typ": "wire:lan",
                    "src": "nid_IoTNetworkInterface_1",
                    "dst": "nid_Switch_1"
                },
                {
```

```
                    "nid": "nid_Switch_1",
                    "typ": "Switch"
                },
                {
                    "nid": "nid_WAN_1",
                    "typ": "WAN"
                },
                {
                    "typ": "wire:external-GW",
                    "src": "nid_ExternalThings_1",
                    "dst": "nid_WAN_1"
                }
            ],
            "rhs": [
                {
                    "nid": "nid_IoTApp_1",
                    "typ": "IoT_App"
                },
                {
                    "typ": "wire:iot_os",
                    "src": "nid_IoTApp_1",
                    "dst": "nid_IoTOS_1"
                },
                {
                    "nid": "nid_IoTOS_1",
                    "typ": "IoT_OS"
                },
                {
                    "typ": "wire:iot_hardware",
                    "src": "nid_IoTOS_1",
                    "dst": "nid_IoTHardwarePlatform_1"
                },
                {
                    "nid": "nid_IoTHardwarePlatform_1",
                    "typ": "IoT_HardwarePlatform"
                },
                {
                    "typ": "wire:iot_network",
                    "src": "nid_IoTHardwarePlatform_1",
                    "dst": "nid_IoTNetworkInterface_1"
                },
                {
                    "nid": "nid_IoTNetworkInterface_1",
                    "typ": "IoT_NetworkInterface"
                },
                {
                    "typ": "wire:lan",
                    "src": "nid_IoTNetworkInterface_1",
                    "dst": "nid_Switch_1"
```

```
                    },
                    {
                        "nid": "nid_Switch_1",
                        "typ": "Switch"
                    },
                    {
                        "typ": "wire:lan",
                        "src": "nid_Router_1",
                        "dst": "nid_Switch_1"
                    },
                    {
                        "nid": "nid_Router_1",
                        "typ": "Router"
                    },
                    {
                        "typ": "wire:WAN-GW",
                        "src": "nid_WAN_1",
                        "dst": "nid_Router_1"
                    },
                    {
                        "nid": "nid_WAN_1",
                        "typ": "WAN"
                    },
                    {
                        "typ": "wire:external-GW",
                        "src": "nid_ExternalThings_1",
                        "dst": "nid_WAN_1"
                    },
                    {
                        "nid": "nid_ExternalThings_1",
                        "typ": "ExternalThings"
                    },
                    {
                        "typ": "MQTT_TLS",
                        "src": "nid_IoTApp_1",
                        "dst": "nid_ExternalThings_1"
                    },
                    {
                        "typ": "IP",
                        "src": "nid_IoTNetworkInterface_1",
                        "dst": "nid_ExternalThings_1"
                    }
                ]
            }
        ]
}
```

Appendix B

Ontology with General Class Axiom Definitions and SMT Verification

```
1  from z3 import * # sudo pip3 install z3-solver
2  import rdflib # pip3 install rdflib
3  import pickle
4  import os
5
6  global g
7  global validation_arg # Z3
8
9  OWL = rdflib.namespace.OWL
10 RDF = rdflib.namespace.RDF
11 XML = rdflib.Namespace("http://www.w3.org/XML/1998/namespace")
12 XSD = rdflib.namespace.XSD
13 DCTERMS = rdflib.namespace.DCTERMS
14 RDFS = rdflib.namespace.RDFS
15 #VANN = rdflib.Namespace("http://purl.org/vocab/vann/")
16 FOAF = rdflib.namespace.FOAF
17 #SAREF = rdflib.Namespace("https://saref.etsi.org/core/")
18 #S4ECHONET = rdflib.Namespace("https://echonet/releaseM/")
19 TEST = rdflib.Namespace("http://quadcube.xyz/ontologies/
       iotvalidation#") # might have some bug with file parsing or
       export from protege. missing '#' in between URI and key
20
21 def dependsOn(char, deps): # Z3 related
22     if is_expr(deps):
23         return Implies(char, deps)
24     else:
25         return And([ Implies(char, dep) for dep in deps ])
26
27 def conflict(*chars): # Z3 related
28     return Or([ Not(char) for char in chars ])
29
30 def check(*problem): # Z3 related
31     s = Solver()
```

```
32      s.set(unsat_core=True)
33      #s.set(':core.minimize', True)
34      for constraint in problem:
35          s.assert_and_track(constraint,str(constraint))
36          #s.add(*problem)
37
38      if s.check() == sat:
39          m = s.model()
40          r = []
41          for x in m: # x is a Z3 declaration
42              if is_true(m[x]):
43                  r.append(x()) # x() returns the Z3 expression
44          print("SAT")
45          print(r)
46          print(m)
47      else:
48          print("unSAT")
49          #print("Proof unSAT")
50          #print(s.proof())
51          print("unSAT core")
52          print(s.unsat_core()) # minimal unsatisfiable core
53
54  def rdf_search(search_subj=None, search_pred=None, search_obj=
        None):
55      global g
56      no_search_keys = 0
57      no_keys_found = 0
58      if search_subj != None:
59          no_search_keys += 1
60      if search_pred != None:
61          no_search_keys += 1
62      if search_obj != None:
63          no_search_keys += 1
64      for subj, pred, obj in g:
65          if subj == search_subj:
66              no_keys_found += 1
67          if pred == search_pred:
68              no_keys_found += 1
69          if obj == search_obj:
70              no_keys_found += 1
71          if no_keys_found == no_search_keys:
72              return (subj, pred, obj)
73          else:
74              no_keys_found = 0
75      return None
76
77  def rdf_axiom_to_z3_builder(search_str):
78      global g
79      global validation_arg
```

```
80      counter = 0
81      search_key = [search_str] # only store latest node ID
82      temp_search_key = []
83      key_property_list = [] # main assumption: keys are all
            serialized in the same order
84      no_constraint_list = [] # main assumption: keys are all
            serialized in the same order
85      constraint_list = [] # main assumption: keys are all
            serialized in the same order
86      while True:
87          for key in search_key:
88              if key != None: # padding mechanism
89                  for subj, pred, obj in g:
90                      if obj == key and pred == RDFS.subClassOf
                            and type(key) != rdflib.term.BNode and
                            type(subj) == rdflib.term.BNode:
91                          temp_search_key.append(subj)
92                      elif subj == key:
93                          if pred == OWL.someValuesFrom: # do we
                                need to check for RDF#type? (Obj:
                                Restriction)
94                              temp_search_key.append(obj)
95                              rdf_search_res = rdf_search(
                                    search_subj=key, search_pred=OWL.
                                    onProperty)
96                              if rdf_search_res != None:
97                                  key_property_list.append(
                                        rdf_search_res[2])
98                          elif pred == OWL.withRestrictions:
99                              temp_search_key.append(obj)
100                         elif pred == RDF.first:
101                             rdf_search_res = rdf_search(
                                    search_subj=obj)
102                             if rdf_search_res != None:
103                                 try:
104                                     print(f'counter: {counter}')
105                                     no_constraint_list[counter]
                                            += 1
106                                 except IndexError:
107                                     no_constraint_list.append(1)
108                                     print(f'no_constraint_list:
                                            {no_constraint_list}')
109                                 try:
110                                     if no_constraint_list[
                                            counter] == 1:
111                                         constraint_list.append
                                                ([(rdf_search_res[1],
                                                rdf_search_res[2])])
112                                     else:
```

```
113                                         constraint_list[counter
                                                ].append((
                                                rdf_search_res[1],
                                                rdf_search_res[2]))
114                                 except: # TODO: find out why
                                        counter increments way more
                                        than what is in the list..
                                        probably have better way to
                                        escape the error
115                                     print(f'Error occured!\
                                            nno_constraint_list: {
                                            no_constraint_list}\
                                            nconstraint_list: {
                                            constraint_list}')
116                             elif pred == RDF.rest and obj != RDF.nil
                                    :
117                                 for _ in range(counter): # padding
                                        mechanism for correct increment
                                        of no. of constraints for a
                                        property
118                                     temp_search_key.append(None)
119                                 temp_search_key.append(obj)
120             counter += 1 # cleaner than list[search_key.index(
                    key)]
121         if temp_search_key == []:
122             break
123         else:
124             counter = 0
125             search_key = temp_search_key
126             temp_search_key = []
127
128     # Z3 constraint builder
129     print("\nZ3 conversion")
130     z3_bool_list = []
131     counter = 0
132     z3_int_list = []
133     z3_int_list.append(Int("{i}".format(i=search_str.toPython().
            split('#')[1])))
134     print(type(search_str.toPython()))
135     for key_property in key_property_list:
136         z3_bool_list.append(Bool("{i}".format(i=key_property.
                toPython().split('#')[1])))
137         print(f'{constraint_list[counter]}\n')
138         for i in range(len(constraint_list[counter])):
139             property_comparison, property_value =
                    constraint_list[counter][i]
140             print(f'\n{property_comparison.toPython()}\t{
                    property_value.toPython()}')
141             if property_comparison == XSD.minInclusive:
```

```
142                     validation_arg.append(If(z3_bool_list[counter]==
                            True, z3_int_list[0] >= property_value.
                            toPython(), True))
143                 elif property_comparison == XSD.minExclusive:
144                     validation_arg.append(If(z3_bool_list[counter]==
                            True, z3_int_list[0] > property_value.
                            toPython(), True))
145                 elif property_comparison == XSD.maxInclusive:
146                     validation_arg.append(If(z3_bool_list[counter]==
                            True, z3_int_list[0] <= property_value.
                            toPython(), True))
147                 elif property_comparison == XSD.maxExclusive:
148                     validation_arg.append(If(z3_bool_list[counter]==
                            True, z3_int_list[0] < property_value.
                            toPython(), True))
149             counter += 1
150
151
152 if os.path.isfile("tmp.pickle"): # load from pickle
153     g = pickle.load(open("tmp.pickle", "rb"))
154 else: # first time load, slow af...
155     g = rdflib.Graph()
156     g.parse("iotvalidation_xml.owl", format="application/rdf+xml
            ")
157     pickle.dump(g, open("tmp.pickle", "wb"))
158
159 validation_arg = [] # Z3
160 search_str = rdflib.URIRef(TEST.E2ELatency)
161 rdf_axiom_to_z3_builder(search_str)
162 validation_arg.append(Int('E2ELatency') == 5)
163 validation_arg.append(Bool('Low') == True)
164 check(*validation_arg)
```