Intent-Driven Secure System Design: Methodology and Implementation

Sian En Ooi[a], Razvan Beuran[a], Takayuki Kuroda[b], Takuya Kuwahara[b], Ryosuke Hotchi[b], Norihito Fujita[b], Yasuo Tan[a]

[a]*Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Nomi, 923-1211, Ishikawa, Japan*
[b]*NEC Corporation, Shiba 5-7-1, Minato-ku, 108-8001, Tokyo, Japan*

Abstract

Given the typical complexity of networked systems in terms of number of components and their interconnections, manually designing their architecture is inherently difficult, and the design process requires expert knowledge and skills. If we also consider the security requirements that networked systems must meet, the task becomes even more demanding, since the manual audit and security mitigation of the architecture are time and labor intensive. This led to research on automated system design, including ways to cover the related security aspects.

In this paper we present a methodology for secure system design that uses an intent-based representation of the network service requirements as input, which is annotated with security requirements, and applies the Design Space Exploration (DSE) approach to generate the system design. Security is handled via a MITRE ATT&CK-based security knowledge base, and a set of security check functions, so that the resulting system design meets not only the functional and quantitative requirements, but also the specified security requirements. We implemented this methodology as the secure system designer SecureWeaver by extending the functionality of an existing intent-based system designer that targeted IT/NW services, named Weaver. A case study of a typical corporate network scenario is used to illustrate the feasibility of the methodology in producing a system design that mitigates the associated security threats. The performance evaluation we conducted for this scenario demonstrates that the added security check overhead does not have a significant impact on the overall performance characteristics of the framework.

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

1. Introduction

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 (Wu et al., 2021). 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 manner in which they should be integrated in order to meet customer needs.

In order to formalize the expression of network service requirements, several classes of approaches have been proposed, such as intent-based representations (Rafiq et al., 2020; Pham and Hoang, 2016; Wu et al., 2021; Kim et al., 2020; Jacobs et al., 2021), and template-based representations (El Houssaini et al., 2015; DesLauriers et al., 2021; Paladi et al., 2018). 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 (Wei et al., 2020). 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 be applied consistently across the network. Typically, a business intent can be either captured through a graphical user interface (GUI) or manually by defining it 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 (Kang, 2016; Pimentel, 2020). 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.

Our work too focuses on extending the DSE approach to take into account system security, and for this purpose we leveraged Weaver (Kuroda et al., 2019; Kuwahara et al., 2021), which is an intent-based system designer that targets IT/NW services. The main difference to other research is that Weaver adds support in DSE for specifying a 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. The approach of Weaver is generic, not being limited to specific domains—such as system-on-a-chip design, or inter-cloud service configuration design, as it happens with other systems—hence we selected it as the basis of our implementation.

However, while Weaver can solve both qualitative and quantitative constraints related to networked system design, it lacks a way to ensure that the designed system is secure. Consequently, the extension that we implemented, named SecureWeaver, adds the necessary functionality to makes it possible to automatically design generic networked systems that are secure, regardless of their specific domains (IT/NW, IoT, etc.). SecureWeaver's intended users are the application experts, and SecureWeaver can help them to design a secure system design for their target application without the direct involvement of networking and security experts. SecureWeaver transforms business requirements or business needs regarding an organization's network into a system design that satisfies the qualitative, quantitative, and security requirements about that network. A business requirement is a high-level description of a business goal or project objective. An example of a business requirement for an IT organization can be simply to provide a cloud storage service to its users. Such business requirements may also contain various non-functional requirements, such as a concurrent number of users, network speed, and many more.

This goal is achieved through the following main contributions presented in this paper:

- Extended the functionality of Weaver to manage internally security-related information, and added security-specific system design rules
- Employed the MITRE ATT&CK taxonomy to build a comprehensive knowledge base that includes both security threats and mitigations, which is used by SecureWeaver to automatically verify system design security characteristics
- Implemented a security check mechanism that checks whether the threats that are present in the service requirement input are mitigated in the system design output

A preliminary version of SecureWeaver was presented in (Ooi et al., 2022). The present paper discusses the following methodology and implementation extensions: (i) the secure design rules and knowledge base were revised to fully include the network domain of the MITRE ATT&CK taxonomy (Section 4); (ii) the security check mechanism implementation was extended to cover all the mitigations in the network domain of MITRE ATT&CK (Section 5); (iii) we employed a realistic evaluation scenario based on a typical corporate network to validate the functionality and performance of SecureWeaver thoroughly assessing all the security check functions (Section 6).

The remainder of this paper is organized as follows. Section 2 introduces related work to our research, such as intent-based design and secure system design. Our methodology for intent-driven secure system design, including implementation requirements and an overview of Weaver, is presented in Section 3. The secure design database that we built based on the MITRE ATT&CK taxonomy and the corresponding security check mechanism are detailed in Sections 4 and 5, respectively. The evaluation of SecureWeaver from functionality and performance perspectives is presented in Section 6. The paper ends with conclusions and references.

2. Related Work

In this section we discuss research related to SecureWeaver and the differences with respect to it.

2.1. *Intent-Based Design*

The work in (Pham and Hoang, 2016) is an intent-based Northbound Interface (NBI) framework for Software-Defined Network (SDN) applications. It translates an application's objectives, requirements and constraints without requiring network-specific language in the intent or comprehension of the underlying network that supports the application in an SDN environment. Although the framework is specific to the SDN domain, the concept behind its intent-based "translation" is similar to SecureWeaver.

The work in (Davoli et al., 2019) is an intent-based NBI framework as well, but for OpenFlow/IoT SDN domains. By using a service chaining description as intent, the framework is able to provision the network path given constraints such as end-to-end Quality-of-Service (QoS) for both the IT/NW and IoT domains. The main difference to SecureWeaver is that it is intended for managing network paths in existing SDN environments, while SecureWeaver is able to design a system and its corresponding network from ground up.

The research presented in (Jacobs et al., 2021) is an intent-based network management framework using natural language. It accepts a natural-language intent from the user and extracts the required information using machine learning algorithm. The required information is then passed to an intent assembly module that generates concrete network configuration commands to meet the network management intent requirements. This work is an intent-based network management tool for existing network infrastructure that does not meet our goal of automatic secure system design.

Existing automatic system design tools (Wu et al., 2021; Rafiq et al., 2020) that can be utilized to design a system are generally either architecture or parameter-level design platforms for their specific domains. Entities and requirements in a topology graph, such as components, relationships and constraints, are specified in an architecture-level design; in parameter-level design, the parameters and fine tuning are specified according to the given topology. Parameter-level design tools such as (Wu et al., 2021) have high flexibility compared to their architectural-level design counterparts. Nevertheless, they are incompatible with our intended use cases, as we focus on architecture-level design for the IT/NW and IoT domains.

2.2. *Secure Intent-Based Design*

The work in (Scheid et al., 2017) is an intent-based designer for Network Function Virtualization (NFV) that models and computes virtual network functions (VNF) chains to meet the intent requirements, including security. It requires definitions of abstract weights for the non-functional requirements in order to perform its quantitative computation. These quantitative scores of VNF are clustered to their respective levels to find the most suitable VNF that satisfies the given requirements. This approach is not as flexible as SecureWeaver for architecture design.

The work in (Amato et al., 2018) is a model-driven design for orchestrated cloud services that includes a security-level evaluation. A template-based matching technique is used to satisfy the service requirements. This research also uses numerical calculations to evaluate the security propagation in the topology, so as to verify whether the output satisfies the required security level. On the other hand, SecureWeaver specifically maps the security threats to their respective mitigations by consulting a threat mitigation knowledge base relying on MITRE ATT&CK data, which is more concrete than an abstract quantitative assessment.

The work in (Kim et al., 2020) is an intent-based security service automation for cloud environments, named IBCS (Intent-Based Cloud Services for Security Applications). IBCS interprets user's network security intent and translates it into concrete configurable security policies. This is done by extracting the data via the Deterministic Finite Automata (DFA) method before mapping it to a suitable Network Security Function (NSF). The Context-Free Grammar (CFG) approach is used to convert the extracted data into security policies, such that the output configuration can be applied to the relevant network interfaces. The difference with SecureWeaver is that IBCS only generates the security policies to be applied to an existing cloud environment, whereas SecureWeaver designs a secure system architecture from ground up that satisfies the given intent.

The work in (Gressl et al., 2021) is a DSE-based system designer for IoT and cyber-physical systems (CPS) applications, using DSE techniques to refine a secure embedded system topology. The main difference to SecureWeaver is that it is specifically used to design the individual properties of IoT device hardware and software, while SecureWeaver addresses networked system architecture design issues. Furthermore, it derives its attack types via Microsoft's STRIDE threat model, whereas SecureWeaver uses MITRE ATT&CK data to define the threats and mitigations.

2.3. *Security Knowledge Framework*

The work in (Kwon et al., 2020) presents a methodology to map the ATT&CK Industrial Control Systems (ICS) matrix into the Facility Cybersecurity Framework

(FCF) cybersecurity assessment tool, as a cyber threat dictionary. The cyber threat dictionary provides mapping between threats and their mitigations that can be used for both proactive and reactive measures. While this work shares the similar concept of using a cybersecurity attack and defense framework for mitigating a threat, SecureWeaver further introduces a security check mechanism to enable the automatic checking of the applied mitigations.

The work in (Hemberg et al., 2020) also presents a methodology to bidirectionally map various 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 work demonstrates the feasibility of expanding the SecureWeaver threat mitigation knowledge base beyond the scope of ATT&CK to include other security databases in the future.

3. Methodology of Automated Secure System Design

This section introduces our methodology for automated secure system design, while the next sections provide further details on the automatic generation of a concrete and secure system design.

3.1. Methodology Overview

During the architecture design phase for a given system, both functional and non-functional requirements, which include the security, safety and privacy of the intended system, must be captured. While safety and privacy are not covered explicitly in this paper, we emphasize that having strong security is a prerequisite for both safety and privacy.

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 (Køien, 2020). 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.

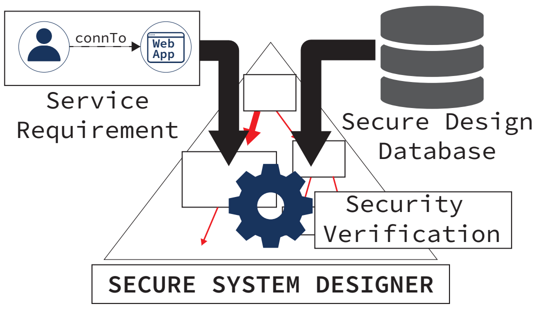


Figure 1: Overview of the secure system designer SecureWeaver.

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 (Køien, 2020).

3.2. Automated Secure System Design Requirements

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 qualitative and quantitative input requirements (see Section 3.3)
2. Use of a secure design database with relevant design patterns and knowledge that make possible the secure system design (see Section 4)
3. Use of a security check mechanism to automatically check that the generated system design satisfies appropriately the input security requirements (see Section 5)

This methodology is illustrated in Fig. 1, 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 check 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 by an application expert is typically a two stage process: (i) capturing the input requirement conceptually, and (ii) composing the service requirement file according to the SecureWeaver format. 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, the application expert will then express these requirements to create the actual content inside the intent file in a format that is compatible with SecureWeaver.

Components that conceptually or concretely describe 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. Hence, we can explicitly assign a security threat to any of the components and relationships in the service requirement which the automated secure system designer must take into account from a security perspective during the refinement process. In practice, application experts will assign such security threats to the relevant components and relationships based on their general security knowledge. We are assuming that application experts have such knowledge about the threat(s) that are relevant to their application, but do not necessarily know how to fully mitigate them.

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 our 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.

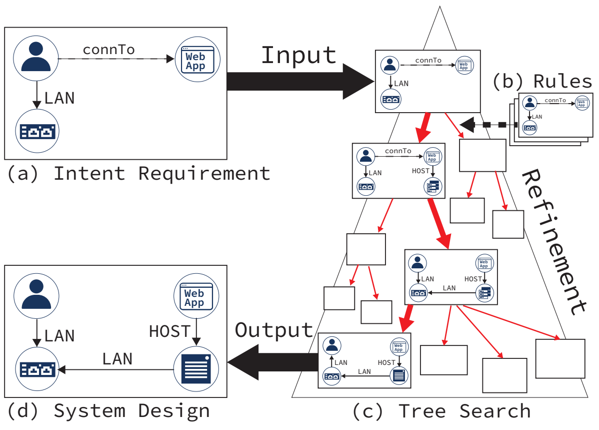


Figure 2: Original Weaver system designer processing flow.

3.3. *Original Weaver System Designer*

Our system design implementation is based on the Weaver IT/NW system designer (Kuroda et al., 2019; Kuwahara et al., 2021). An overview of Weaver's processing flow is presented in Fig. 2. 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 as a generic OS or network connection). The 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 said to be concrete, is then output by Weaver as a system design for the given service requirement. Weaver's data format, rules and topology refinement and tree search approach are briefly described in the following subsections.

3.3.1. *Data Format Definitions*

Weaver's data format is structured closely to TOSCA (Topology and Orchestration Specification for Cloud Applications) (Rutkowski et al., 2020), a specification that declaratively describes 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 relationship 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.

3.3.2. *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 rules. 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 a rule. An illustration of a rule, `DEPLOY-APP`, is shown in Fig. 3. 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. 3 is transformed into t_2 by the rule `DEPLOY-APP` with the match pattern, $m : \{1\} \mapsto wa$. The arrow with dashed line represents an abstract relationship, while an arrow with solid line represents a concrete relationship. The base component App in Fig. 3 has an inheritance relationship with the $WebApp$ component type. Hence, component $\{1\}$ of type App is

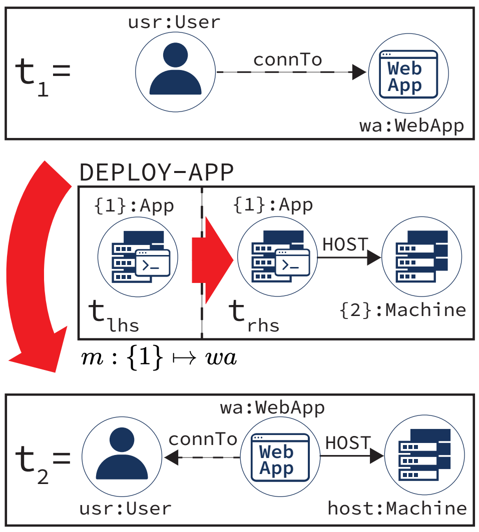


Figure 3: Example of the application of the rule `DEPLOY-APP` as a Weaver topology refinement step.

equivalent to wa of type $WebApp$, such that $\{1\}$ in the rule `DEPLOY-APP` can be replaced by $\{1\} \xrightarrow{HOST} \{2\}$.

3.3.3. *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 entities in the topology are fully concretized; for detailed information on the tree search algorithm and the definition of topology concreteness, see (Kuwahara et al., 2021).

4. Secure Design Database

In this section we introduce the secure design database of SecureWeaver that includes: (i) security threat information applicable to system design, and refinement rules that are compatible with our threat mitigation approach; (ii) security knowledge from third-party security databases that pertains to secure design.

4.1. *Secure Design Threats and Rules*

We present here the first component of the SecureWeaver database, the security threats used for secure system design, including the concept of logical and conceptual connections introduced to extend the existing Weaver capabilities, and the refinement rules that we defined, illustrating with examples the various types of rules and corresponding use-case scenarios.

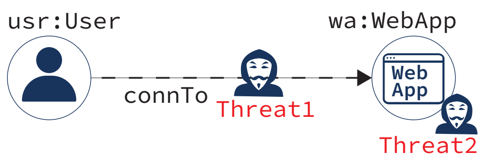


Figure 4: Service requirement that includes relationship-type and component-type threats.

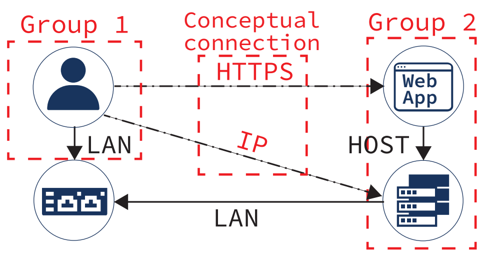


Figure 5: Logical and conceptual connections between two system component groups.

4.1.1. Security Threats

Security threats are of two types: (i) component-type threats that are applicable to system components, such as a physical host; (ii) relationship-type threats that pertain to relationships between such components, such as the network connection between a device and an application. In the first case the threat indicates the target component requires protection, and in the second case it is the attack path to a target that requires protection.

In Fig. 4 we show an example of a service requirement that includes an explicitly-defined relationship-type threat, `Threat1`, applied to the `connTo` connection between the user `usr` and the web application `wa`, and a component-type threat, `Threat2`, applied to `wa`. The relationship threat signifies in this case that the connection is susceptible to sniffing or eavesdropping that would affect the security and privacy of the two end components while the component threat signifies that the component is susceptible to a certain malicious activity. The threat in this and future examples is represented by an Anonymous mask icon placed on the affected relationship/component.

Modeling component-type threats is relatively straightforward, as this type of threat only affects the target component and any sub-components derived from it in the design process. However, a relationship-type threat affects multiple components and sub-components in a topology. Therefore, in order to model relationship-type threats and their inheritance for the affected branches in a topology, the root relationship between two components (e.g., `connTo` between `usr` and `wa` in Fig. 4) has to be preserved as the topology is refined, so as to make it possible to verify possible mitigations. For this purpose, the existing Weaver functionality was extended to accommodate such information, which we achieved by introducing two new types of connections, logical and conceptual, as explained next.

4.1.2. Logical and Conceptual Connections

Logical connections are defined as relationships with an *rtype* property that corresponds to application and network layer protocols in the TCP/IP model. The values of logical connections *rtype* are concrete, such as `HTTP`, `HTTPS`, `RTP`, `SRTP`, `IPSEC` and `IP`. Fig. 5 includes two such concrete logical connections, `HTTPS` and `IP`, which are represented by dash-dot lines. Note that the TCP/IP model is well suited for describing the relationship between two network components, and considering only the application and network layers is currently sufficient for our system design purposes.

A conceptual connection is defined as a pair of application and network layer *rtype* objects that connects two component groups in a topology. A component group means a set of one or more components that were derived via refinement rules from a given component in the service requirement, and that corresponds logically to a service specified in that requirement. Fig. 5 includes two such groups, one made of the user, which forms a group by itself, and another made of the web application and the host it is deployed on, as the latter was added to the topology via a refinement rule corresponding to the web application.

The conceptual connection between these two groups is the pair of the `HTTPS` and `IP` logical connections. Retaining such logical and conceptual connections in the topology preserves the information needed by the subsequent security check algorithm (see Section 5).

4.1.3. Refinement Rules

The refinement rules in the secure design database refer to: (i) component refinement; (ii) relationship refinement. These secure design rules are in addition to the generic refinement rules for both components and relationships that are needed to design those aspects of the system that are not security related.

A graphical representation of security-related refinement rules is shown in Fig. 6. Refinement rules such as `DEPLOY-NIDS` and `DEPLOY-FIREWALL` illustrate the type of rules that add network security appliances into a topology. There are also rules that refine an abstract component into a concrete type for security purposes,

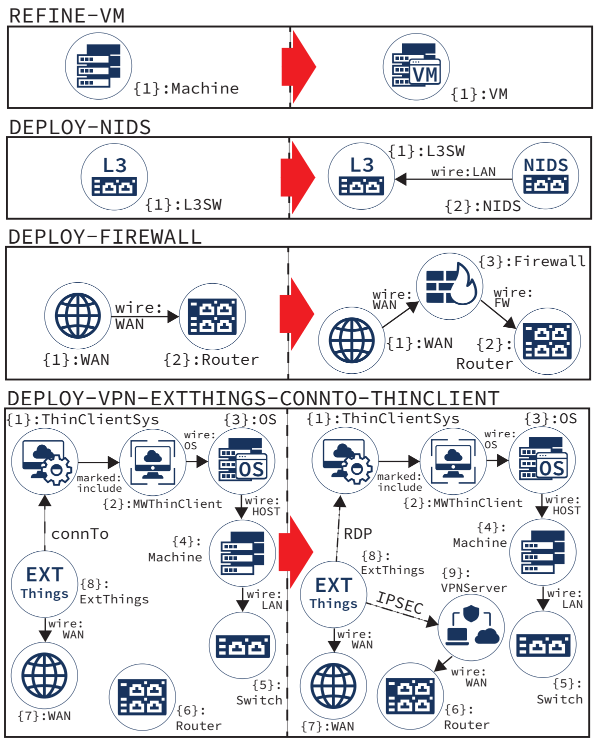


Figure 6: Graphical representation of security-related refinement rules.

such as REFINE-VM that refines an abstract Machine type component into a VirtualMachine instead of the regular PhysicalMachine or PhysicalServer, so as to provide isolation or a sandboxed environment.

Next we look at the refinement rules for the refinement of the relationship type, *rtype*. Refinement rules in Weaver transform one topology state to another by removing or changing the abstract *ctype* or *rtype* entities to achieve a concrete state. On the other hand, for the concrete logical connections in SecureWeaver, the refinement rules have to be designed in a way that ensures that logical connections are preserved in the topology.

Note that when it comes to security concerns in refinement rules, top-level threats are explicitly included in the service requirement. However, protocol-related threats and mitigations are implicitly included through the inherent security properties of TCP/IP application and network layer protocols when used as an *rtype* in a topology; consequently, they are defined in the threat mitigation knowledge base, and used for security checking purposes.

Refinement Mechanism. During topology refinement, there may be more than one solution to mitigate a threat. Fig. 7 illustrates a service requirement for which at time

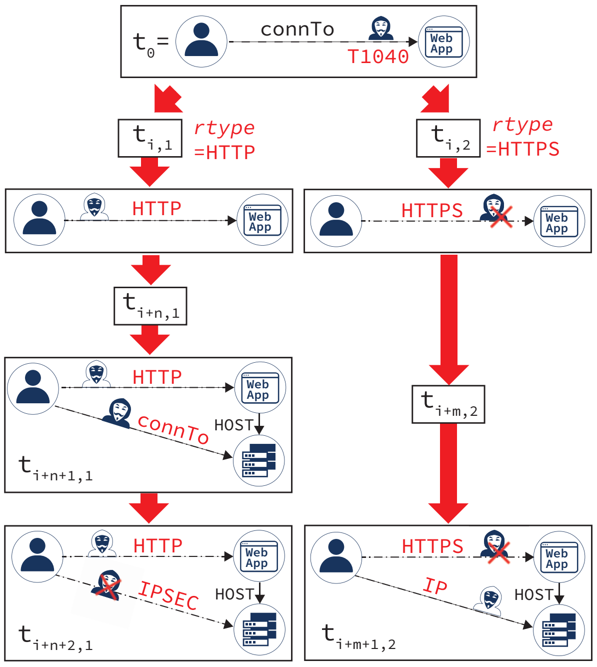


Figure 7: Example of possible topology refinements that mitigate the security threat T1040 defined in service requirement via the use of security protocols (mitigation M1041).

t_0 a User and a WebApp application are connected by the relationship connTo to which the threat T1040 is applied. The relationship connTo can be concretized by using either the HTTP or HTTPS application protocols, where HTTP is insecure and HTTPS is secure. The left and right-hand side diagrams in Fig. 7 show the sequential topology refinements for the HTTP and HTTPS branches, respectively.

For the left-hand side topology, HTTP does not mitigate the threat T1040, and the associated implicit threat on the affected relationship is denoted by a white Anonymous mask icon. Following the refinement procedure of this topology, the implicit threat is inherited by the connTo abstract *rtype* relationship as shown at step $t_{i+n+1,1}$. Inheritance in this context means that if the application layer relationship has an implicit threat, the same threat is also applied implicitly to the network layer relationship. At time $t_{i+n+2,1}$, the abstract connTo relationship is replaced with a concrete IPSEC logical connection. At this point, the T1040 threat is mitigated, as IPSEC is defined as a valid mitigation for this threat in the threat mitigation knowledge base. Note that we denote mitigated threats by an Anonymous mask icon marked with a red “X”.

For the right-hand side topology, HTTPS mitigates the T1040 threat, as per the definition in the threat mitigation knowledge base. Hence, the remaining refinement for concretizing this topology does not have any security issues, and the insecure IP protocol can be used at network level without affecting the overall security characteristics of the solution.

The above discussion about the left-hand side topology demonstrates that logical connections that do not mitigate a threat should not be eliminated immediately, as it could truncate potential solutions. Therefore, for SecureWeaver we only perform the security check after all the entities in the topology are fully concretized.

4.2. *Threat Mitigation Knowledge Base*

There are various methodologies for security assessment, such as Lockheed Martin Cyber Kill Chain (CKC) (Martin, 2014), MITRE ATT&CK (Strom et al., 2018), and Microsoft STRIDE (Hernan et al., 2006). CKC is a well-known intrusion-centric framework that describes a well-defined sequence of attack steps. ATT&CK, on the other hand, is a list of attack techniques grouped by tactics that does not imply any specific order of operation. STRIDE is a high-level threat model, commonly used during the security development life-cycle, as it is focused on identifying overall categories of attacks.

Since for SecureWeaver threats are defined explicitly in the service requirement, the kill chain and high-level modelling techniques are not suitable, as they are incompatible with the Weaver refinement process, and do not include sufficient security context. Therefore, for our threat mitigation knowledge base, we selected the ATT&CK framework as it provides a rich taxonomy of adversarial tactics, techniques, and common knowledge that can be readily used in various scenarios.

Note that other third-party security databases besides ATT&CK should be compatible for use in tandem with ATT&CK in the threat and mitigation knowledge base. This is achievable via the usage of the common cyber threat intelligence (CTI) sharing language, Structured Threat Information Expression (STIX) (Barnum, 2012).

4.2.1. *MITRE ATT&CK-based Mitigation*

The ATT&CK knowledge base is organized as a collection of matrices, such as Enterprise, Mobile and Industrial Control Systems (ICS), with each matrix covering a different field of adversary tactics, techniques, and procedures (TTP). The data in an ATT&CK matrix is categorised into attack tactics, techniques, sub-techniques, and mitigations. For our knowledge base, the most relevant components are threats and mitigations. Therefore, we included the ATT&CK attack techniques and mitigations as the security threats and mitigations in our knowledge base.

The ATT&CK version used in our threat mitigation knowledge base is v10, which is the latest version at the time of writing. Specifically, we focus on the ATT&CK Enterprise matrix, which contains a total of 14 tactics, 185 techniques, 367 sub-techniques and 42 mitigations.

Since the total number of techniques and sub-techniques in ATT&CK is large, we also leveraged the concept of "domain" that is used to group relevant techniques and sub-techniques based on their area of relevance. There are currently seven domains in the ATT&CK Enterprise matrix: (i) Preparatory; (ii) Windows; (iii) macOS; (iv) Linux; (v) Cloud; (vi) Network; and (vii) Containers.

The network domain in ATT&CK is the most relevant to SecureWeaver, given our focus on networked systems. The network domain includes 9 tactics, 15 techniques, 26 mitigations and 16 data sources, some of them unique and some of them shared with other domains. Table 1 shows the threats, mitigations and the affected Weaver entity types for the network domain. Threats are listed in the order displayed on the MITRE ATT&CK Web page (`https://attack.mitre.org`). Note that for some techniques (e.g., `T1600`, `T1056`) no known mitigation exists in ATT&CK, and we denoted this by "N/A" in the table. The affected Weaver entity types were assigned after analyzing the attack surface/vector for each threat. For the meaning of the mitigation IDs, see Table 2.

By analyzing the attack techniques and mitigations in the ATT&CK network domain we obtained the necessary information needed to implement SecureWeaver functions that can verify whether a certain mitigation is applied for a given system topology. Through this analysis we have identified seven types of verifiable characteristics, as follows:

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 also shown via notes in Table 1; the actual security check functions, which provide full coverage for the MITRE ATT&CK network domain, will be presented in detail in Section 5.

Table 1: MITRE ATT&CK threats and mitigations for the 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

4.2.2. Knowledge Base Data Structure

For the data structure of the threat mitigation knowledge base, we implemented a JSON key-value pair representation to store the relevant information, which is organized in three main categories: (i) weight of the security mitigation; (ii) mitigation check function mapping; (iii) secure protocol corresponding to the threat and mitigation pair (if any).

To illustrate the mitigation weight concept that we introduced, we will use the ATT&CK Exploit Public-Facing Application (`T1190`) threat as an example. As shown in Table 1, `T1190` has six possible mitigations, which are sandboxing mitigation, firewall mitigation, network segmentation, and three configuration related mitigations. Each mitigation may help achieve a certain security level against the target threat, and we assume that if all the mitigations are applied, the threat is fully neutralized. Hence, each mitigation may have a different weight assigned to it, depending on its significance, with the total sum of the weights for all the mitigations corresponding to a threat being equal to 1.0. While these weights can be given equal values as a default solution, we envision that expert knowledge can be used to assign more realistic values.

Each threat and mitigation pair is mapped to the corresponding security check function introduced in Section 4.2.1, along with the relevant Weaver object type (Component/Relationship) to which the mitigation is to be applied. For the particular case of configuration settings check, we use the knowledge base to assign `True` or `False` values, depending on whether the corresponding system settings are assumed to have been enacted or not. If Weaver gains the functionality of checking the actual settings in the future, then this aspect can be verified dynamically as well.

Information about the logical connections introduced in Section 4.1.2 is also included in the threat mitigation knowledge base. Thus, we define a set of secure network protocols, such as `HTTPS` and `IPSEC`, and map them to the corresponding threat and mitigation pair in the knowledge base. Any logical connection types that are not explicitly defined in the knowledge base are deemed to be insecure.

5. Security Check Mechanism

In this section, we introduce the security check mechanism and functions in SecureWeaver. Details of the security check mechanism are presented first, followed by

Table 2: 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

the seven security check functions introduced in Section 4.2.1 that provide full security check coverage of the network domain in the ATT&CK framework.

5.1. Mechanism Overview

To address the issue of threat mitigation, we implemented a security check algorithm that ensures every threat specified in the service requirement is mitigated before Weaver can finalize the system design. Currently, only the threats that are explicitly defined in the service requirement are considered, and a system design is deemed secure only if all those threats are mitigated according to the desired security level policy.

The security check takes place after SecureWeaver checks that the system design contains no abstract entities, and is divided into two stages: (i) retrieving the threats from the service requirement, and (ii) calling the appropriate security check functions. Next we explain this process in detail, followed by a description of each security check function in Sections 5.2 through 5.8.

5.1.1. Retrieving Threats from the Service Requirement

The first part of the security check process searches for all the component and relationship type threats, n_{threat} and e_{threat}, present in the service requirement input, t_0, and stores them into their respective threat arrays. Each threat in those arrays is individually checked from a mitigation perspective, unless the security check process is explicitly marked to be omitted in t_0. The threats to be actually checked are passed to the second part of the security check process.

For each threat in the service requirement input, how to decide whether a threat is mitigated or not must also be specified. This is because there may be more than one mitigation for a particular threat, and the way to check whether that threat was neutralized or not needs to be determined. SecureWeaver implements three security check modes: (i) OR type; (ii) AND type; (iii) based on a configurable security level.

To implement this concept, each threat-mitigation pair is assigned a weight that it used to signify how much of a threat a certain mitigation method mitigates. The OR security check mode means that at least one mitigation should be satisfied for the threat to be neutralized, hence it is equivalent to a minimum security level (this is implemented by checking against a threshold equal to the minimum value of `float` numbers). The AND security check mode means that all mitigations related to the target threat must be satisfied for that threat to be neutralized, being equivalent to a maximum security level (this is implemented by checking against a threshold equal to 1.0, which is the maximum value).

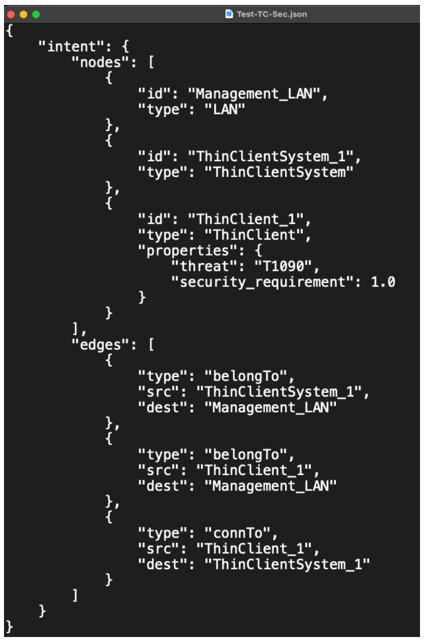
The configurable security level check mode means that a threat is considered to be neutralized if the total calculated security level weight for all the security check functions associated to that threat is larger or equal than the configured security level, which is between 0.0 and 1.0. Actually, to disable the security check for a certain threat, its configurable security level can be set to the value 0.0.

An example of a service requirement with a threat is shown in Fig. 8, where the ATT&CK "Proxy" (`T1090`) threat is defined with the maximum security level 1.0 for the `ThinClient` type component under "properties".

5.1.2. Calling Security Check Functions

The second part of the security check process first determines the security check functions corresponding to the specified threat, then executes them accordingly.

To determine the appropriate security check functions that should be executed, the algorithm will retrieve from the secure design database all the mitigations applicable



```
{
    "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 8: Service requirement example that includes the threat T1090.

to the given threat, and store all the relevant information about the security check functions that correspond to those mitigations in an array. It is also possible to specify explicitly what mitigation to be considered, and then SecureWeaver will retrieve only the security check function that is linked to the provided threat and mitigation pair.

Once the security check function retrieval is complete, the algorithm will check whether there is any object in the array. If the array is empty, meaning that no known security check function could be determined, the algorithm will return `False`, thus rejecting the selected topology state from a security check point of view.

If the array is not empty, each of the threat-mitigation pairs will be first parsed, since component and relationship type security check functions require different sets of arguments. Once the type is determined, the algorithm will dynamically call the target security check function via its name, and pass it all the required arguments.

If the security check function returns `True`, the algorithm will retrieve the security level weight assigned to that specific threat-mitigation pair, and sum it up to the current total weight for the target threat. If the returned value is `False`, then the sum is not updated. After processing all the security check functions in the array, the total weight is rounded off to two decimal places to prevent numerical calculation errors for the AND security check mode.

In the end, the algorithm checks whether the total weight for the target threat is larger or equal to the threshold mentioned in Section 5.1.1, depending on the security check mode (OR, AND or security level-based). If this condition is met, the second part of the security check process will return `True`, and the corresponding threat is considered mitigated. On the other hand, if the security check fails, the selected topology state will be discarded, and the security check process is repeated for the next topology state produced by Weaver.

5.2. Application Isolation and Sandboxing Check

The "Application Isolation and Sandboxing" (M1048) mitigation in the ATT&CK framework aims to restrict execution of code to a virtual environment on or in transit to an endpoint system. In SecureWeaver, we can verify application isolation and sandboxing by checking the existence of a concrete component type such as `VirtualMachine` in the design.

We implemented the isolation/sandboxing check function by calling a more generic subroutine, named `verify_type()`, that checks whether a node of a given type is found while traversing the topology from the source node to which the threat is applied. In particular, this subroutine (see Algorithm 1) verifies whether the node of the desired type, n_{type}, is a child of the input source component, n_{src}.

First, the subroutine retrieves all the relationships from the input source component to other components, and stores them into the array $n_{conn}[]$ as shown on line 2 of Algorithm 1. For each component relationship n_{conn} in $n_{conn}[]$ it then checks whether the component relationship type has the prefix "wire", denoting an internal Weaver connection. If the condition is satisfied, the destination component of n_{conn} is retrieved and assigned to n_{dst}. Then, the n_{dst} type, $n_{dst,type}$, is checked, and if it is the same with n_{type} (Line 6), the subroutine will return `True`, meaning that the searched component type is indeed a child of the input source component. Python "in" syntax is used in this paper where it is used to verify if a value is present in a sequence such as a list or string.

If the check fails, the type check subroutine is called recursively with the component relationship destination, $n_{conn,dst}$, as the new input source component by following all internal Weaver connections except "wire:lan", which is a particular case representing an external LAN connection. In case the searched type is not found and there are no connections left in the $n_{conn}[]$ array, the subroutine will return `False`.

Algorithm 1 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:     $n_{conn}[] \leftarrow$ all connected edges to $n_{src}$
3:     for all $n_{conn}$ do
4:         if "wire:" in $n_{conn,type}$ then
5:             $n_{dst} \leftarrow$ 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
```

In the application isolation and sandboxing check function, the subroutine `verify_type()` discussed above is called for each target node with `VirtualMachine` as node type argument.

5.3. Firewall Use Check

The firewall use check function deals with mitigations such as "Exploit Protection" (`M1050`) for public-facing application threat `T1190` from the ATT&CK framework. According to `M1050`, in order to protect a system against exploits of a public-facing component such as `WebApp`, a firewall or equivalent must be present. The firewall use check algorithm is implemented as two steps: (i) search for child-of target network component, and (ii) network-specific mitigation check.

The child-of target (for network) component search function `search_nw_node()` shown in Algorithm 2 is mostly similar to the type check introduced in Algorithm 1. The main difference is that Algorithm 1 only searches within a single physical machine before $wire : lan$ relationship type, while Algorithm 2 search beyond the $wire : lan$ until it finds a valid type of network component. For each $wire : lan$ that is found in Algorithm 2 line 8, the search algorithm will retrieve the destination object of that relationship to check whether it is a valid type of network component (e.g.: $L3SW$ and $L2SW$). If a valid component type is found, the selected relationship will be stored in an array ($connected\_nw[]$) as shown in Algorithm 2 line 10. After that, the security check function will search exhaustively for the remaining network type relationships and append the selected relationships into the array. If the component is not the target component types, the algorithm will recurse with the destination node of the relationship to find the next child component. When all the relationships in $n_{conn}[]$ have been processed, the algorithm will check array $connected\_nw[]$. If the array is not empty, the algorithm will return the array denoting a successful search. If the array is empty, the algorithm will return `False`, which signifies that the child-of target node is not connected to any network.

Algorithm 2 Search and verify the child-of a target component for network related check.

```
Input: Topology, $t$; Source node, $n_{src}$
Output: True or False
1: function SEARCH_NW_NODE($t, n_{src}$)
2:     $n_{conn}[] \leftarrow$ 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 verify_nw_node($t$, $n_{conn,dst}$)
7:             end if
8:         else
9:             $n_{NW} \leftarrow$ node $n_{conn,dst}$
10:            if $n_{NW,type}$ is a valid switch type then
11:                $connected\_nw[] \leftarrow n_{conn}$
12:            else
13:                return verify_nw_node($t$, $n_{conn,dst}$)
14:            end if
15:        end if
16:    end for
17:    if $connected\_nw[]! = \emptyset$ then
18:        return $connected\_nw[]$
19:    else
20:        return False
21:    end if
22: end function
```

After the first part of the search process is successful, the result is passed into the security check function `verify_nw_node_type()`, which verifies the mitigation based on the type of the network component (see Algorithm 3). The function first iterates through each network relationship, $connected\_nw$ in the array returned by Algorithm 2, where the all the relationship of each network relationship destination, e_{NW} are retrieved. Each of the relationships that are stored in e_{NW} are processed and the object of the relationship source is retrieved as shown in Line 4 to 5. The type of e_{NW} is then checked whether it is in the n_{type} input as shown

Algorithm 3 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
function VERIFY_NW_NODE_TYPE($t$, $connected\_nw[]$, $n_{type}$)
    for all $connected\_nw[]$ do
        $e_{NW}$ ← all connected edges to $connected\_nw_{dst}$
        for all $e_{NW}$ do
            $n_{NW}$ ← node $e_{NW,src}$
            if $n_{NW,typ}$ in $n_{type}$ then
                return True
            else
                $connected\_NW1[]$ ← all connected edges to $e_{NW,src}$
                if verify_nw_node_type($t$, $connected\_NW1[]$, $n_{type}$) then
                    return True
                end if
            end if
        end for
    end for
    return False
end function
```

in Line 6. This allows the algorithm to verify it against both single type or a list of types. If the condition is satisfied, the algorithm will return `True` denoting that the target type is a part of the affected component network. If the condition is not satisfied, the algorithm will retrieve all relationship related to the source of e_{NW} and store it into an array, $connected\_NW1[]$. Then, the algorithm will recurse with $connected\_NW1[]$ provided as the new input array to be verified.

Hence to verify firewall type component, a wrapper function is implemented, which first calls the Algorithm 2 subroutine to obtain the networks the target component is connected to. Then, the wrapper function calls the Algorithm 3 subroutine with the result of Algorithm 2 as input to verify whether a firewall type component is a part of the target component network.

5.4. Network Segmentation check

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.

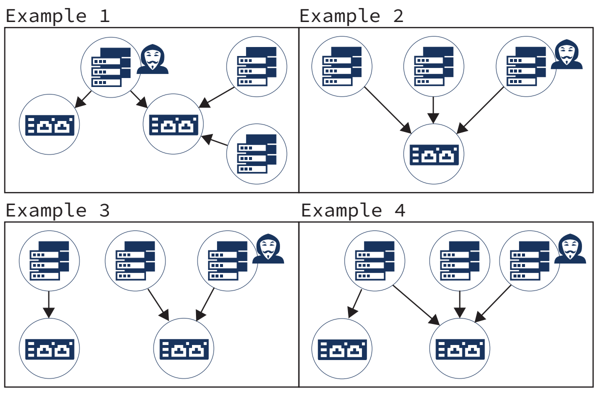


Figure 9: Examples of possible topology states for network segmentation check.

The conditions we used to verify whether a component is segmented or not from a network point of view are shown in Fig. 9. 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 we decide that the component under threat is in 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. 9 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. In Example 4 we show 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 check algorithm a wrapper function was implemented, which calls first the Algorithm 2 subroutine to determine which networks the input source component is connected to. The array of connected networks returned by this subroutine 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. 9.

5.5. *Configuration Settings Check*

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 check 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.

5.6. *Traffic Filtering Check*

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 check 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 software firewall and its configuration settings, the security check function must check the assumed configuration settings in the secure design database. Hence, this requires the security check function to choose or combine the network type check introduced in Section 5.3, and the configuration settings check discussed in 5.5 to determine whether a mitigation is suitable for a particular threat.

We implemented the corresponding security check function as a wrapper function that calls both the aforementioned security check functions. The wrapper function first checks the secure design database for the type of method: (i) hardware; (ii) software; (iii) require either one (OR); and (iv) require both (AND). Then the wrapper function calls the related check 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 check 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 check function will return `True`.

5.7. *Secure Protocol Use Check*

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, we introduced the logical and conceptual connections 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, we implemented a secure protocol check function that checks the security of the application layer and network layer protocols in a given topology state. The security check function, shown in Algorithm 4, 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 the 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.

Algorithm 4 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 $CC_{mitigation}$ != $\emptyset$ then
12:                     return True
13:                 end if
14:             else
15:                 $CC_{mitigation}$ ← $e_{conn}$ mitigation info
16:                 if $CC_{mitigation,threat}$ == $e_{threat}$ and $CC_{mitigation}$ != $\emptyset$ then
17:                     return True
18:                 end if
19:             end if
20:         end if
21:     end for
22:     return False
23: end function
```

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 check function will return `True`, since only secure protocols are recorded in the knowledge base.

5.8. Intrusion Detection and Prevention System (IDPS) Use Check

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, we implemented a wrapper function that is used to search for `NIDS` or `IDPS` type components. This function calls the two subroutines shown in Algorithm 2 and Algorithm 3, 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 check function will return `True`.

6. System Evaluation

In this section we present the evaluation of the SecureWeaver system. The evaluation is first done from a functionality perspective. Thus, we use SecureWeaver to generate a secure system design based on a service requirement input that includes several security threats, and verify that those threats are mitigated in the concrete system topologies that SecureWeaver outputs. For the same scenario we also conduct a performance evaluation to determine the overhead with respect to the Weaver baseline introduced by the additional security check mechanism in SecureWeaver. Finally, we compare the features of SecureWeaver to those of several related systems, highlighting their respective advantages and disadvantages.

SecureWeaver was implemented in Python 3 and is based on Weaver v0.1.3. All the experiments presented in this paper were performed using an Amazon Web

Services (AWS) Elastic Compute Cloud (EC2) VM instance. The EC2 instance utilized is “p2.xlarge,” which features 4 virtual Intel Xeon E5-2686 v4 CPUs with a frequency of up to 3.0 GHz, 64 GB of RAM, an NVIDIA Tesla K80 GPU, and 60 GB of available storage.

6.1. *Functionality Evaluation*

For the functionality evaluation we use service requirement input that is based on a realistic corporate network scenario. After introducing the input scenarios, we then validate the system design output from a security perspective to demonstrate that SecureWeaver is able to generate secure system designs.

6.1.1. *Service Requirement Input*

The service requirement input in our experiments is built using a subset of Weaver components and relationships that are available in the SecureWeaver system model database, as shown in Fig. 10. The light-grey background signifies a group of abstract and concrete components that are related to each other; for example, `System` is an abstract component, while `BackUpSys`, `WebSys`, and `ThinClientSys` are the concrete representations of that system. The other included groups are `WebApp`, related to web applications, `MiddleWare`, consisting of various middleware software applications for backup, thin client and web application server, and `Storage`, which has a Storage Area Network (SAN) system as a concrete component.

Fig. 10 depicts several other groups of system components, such as `Machine` that consists of a hardware or a virtualized host, `ExtThings` that consists of external entities, such as users (`User`) and external API (`ExtAPI`), `LAN` that is related to Ethernet network switches, and the `OS` group, which consists of operating system-related components.

Several independent components that are used in the evaluation are also shown in Fig. 10, such as the Internet or Wide Area Network (`WAN`), a thin client (`ThinClient`), a network router (`Router`), an NIDS appliance (`NIDS`), a VPN server instance (`VPNServer`), and a firewall appliance (`Firewall`). As for the `Requirement` component shown in the same figure, it is used to represent functional or non-functional requirements that are part of components related to `System`.

Using all these components, one can design a typical corporate network that includes various network systems, such as thin clients, remote access, and web application hosting. Three basic scenarios are first introduced, where each scenario has one ATT&CK threat.

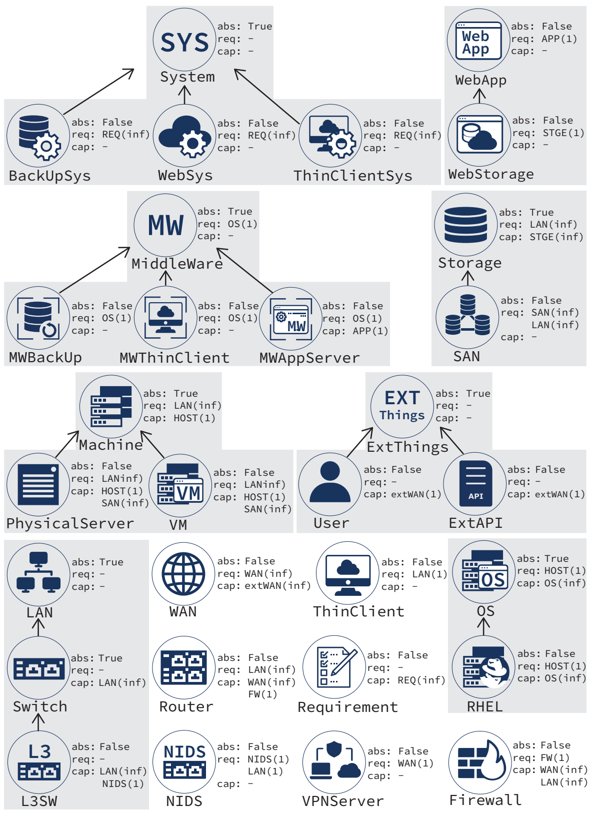


Figure 10: Properties and relationships of the components used in the system evaluation experiments.

The threats were selected as being sufficient for demonstrating the use and validate all the seven security check functions in SecureWeaver. Furthermore, each of these scenarios can be integrated with others to create complex scenarios.

The scenarios below are introduced in the order of their complexity to design them in SecureWeaver. To evaluate SecureWeaver capability to design secure system for real world application, the complete service requirement built up from the three scenarios are shown in Fig. 11. Each of the building block scenarios are grouped using different shades of grey background and are numbered according to the description below.

Scenario #1. A service requirement with a thin client system is illustrated in Fig. 11 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` belong to `bizLAN` LAN network, where both `TC` and `TCSys` 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.

Scenario #2. Often employees need to remotely connect to a thin client environment on-premise to access software that are 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. 11, 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 thin client system in the service requirement is the same with the Scenario #1 service requirement.

Scenario #3. A service requirement with a public-facing web application is illustrated in Scenario #3 in Fig. 11. In this scenario we assume a company has a private-cloud storage system on-premise with the cloud storage accessible via public Internet, and uses some external API for single sign-on (SSO) purposes. These are illustrated in Fig. 11 as the `connTo` relationship between the `WebStorage` type component and both external API (`API`) and external user (`Usr`). Both `Usr` and `API` are connected to the `WAN`.

Moreover, we also introduce a backup system as the functional requirement supporting the private-cloud storage web system (`WSSys`). The web storage application `WS` is a part of `WSSys` as connected via the abstract `include` relationship. Both `WS` and the backup system component `BKSys` are part of the `dmzLAN` as connected with the abstract `belongTo` relationship. In this scenario the "Exploit Public-Facing Application" (T1190) threat is defined for the `WebStorage` component, to model the fact that `WS` services are at risk of public-facing exploitation of zero-day vulnerabilities.

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.

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. 11, with the thin client system being affected by threat T1090, the remote access between the user and the thin client system being affected by threat T1040, and the web storage application being affected by threat T1190.

6.1.2. Security Validation of System Design Output

With the service requirement from the evaluation scenarios, SecureWeaver can generate a concrete and secure system design that corresponds to the selected service requirement. A set of refinement rules that is made of 33 rules was defined and used throughout the evaluation, including the rules illustrated in Fig. 6. In this evaluation we assume that every available mitigations for a threat is of equal effectiveness in mitigating the threat. Thus, the security weights for each mitigation for a threat are evenly divided, with their sum being 1.0.

There are two modes of operation in SecureWeaver: automatic and interactive. In automatic mode, SecureWeaver will by default match the first refinement rule that meets both quantitative and qualitative requirements, and heuristically apply them to generate the next possible topology states. This process is deterministic

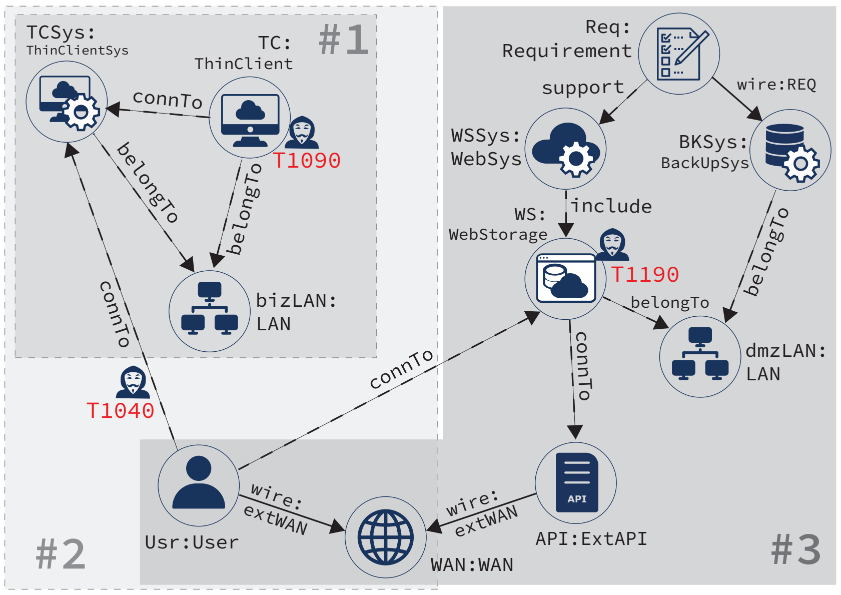


Figure 11: Input service requirement: thin client system with threats T1090 and T1040, and web system with threat T1190.

and SecureWeaver will only return the first system design that satisfies the quantitative, qualitative and security requirements. In interactive mode, SecureWeaver presents all the valid refinement rules that can be applied to the current topology state, and the user can manually select the refinement rule to be applied at each 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 check 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. 12. As mentioned previously, the full scenario is a combination of Scenarios #1, #2 and #3, which are emphasized using different shades of grey background.

The refined Scenario #1 part in Fig. 12 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`), which is denoted by the `L3SW` component type. 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 checks such as traffic filtering check (M1037), IDPS check (M1031), and configuration settings check (M1020) were performed by SecureWeaver.

The traffic filtering check requires that the `TC` has either software firewall or 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. 12. The IDPS check 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` Layer 3 network switch, thus satisfying the requirement.

A NIDS is generally not an effective security mechanism in the case of encrypted traffic. Nevertheless, in our particular scenario the traffic from the Thin Client (TC) is not encrypted, hence the NIDS mitigation (M1031) in MITRE ATT&CK is functional. Note, however, that MITRE ATT&CK does not explicitly differentiate between encrypted and unencrypted traffic for the use of NIDS, and even if the traffic would be encrypted it would not create a conflict with the MITRE framework that we are following. In addition, MITRE ATT&CK includes several mitigations for the threat Proxy T1090, and all of them will be applied when

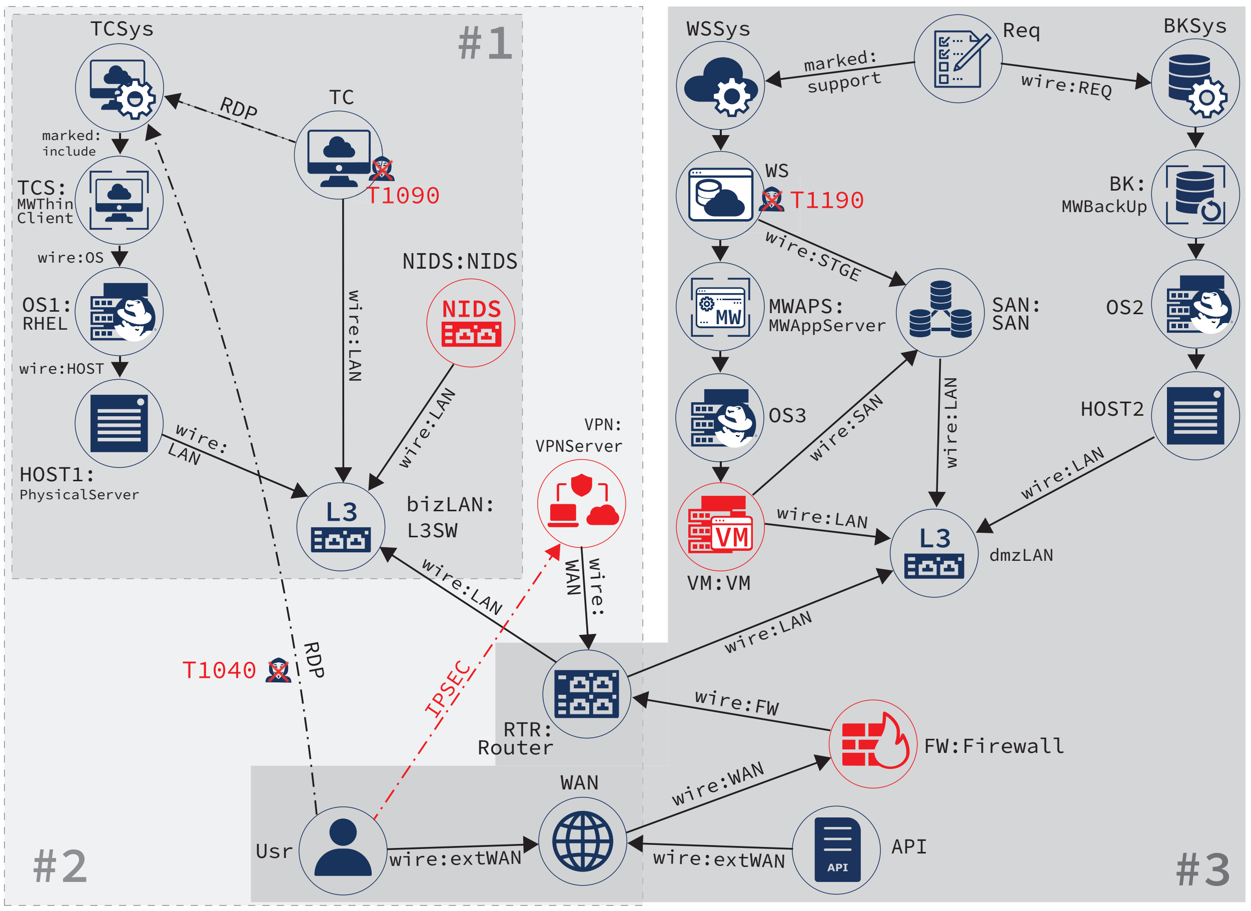


Figure 12: SecureWeaver output system design: refined thin client and web systems with mitigated threats.

the security level is maximum. Specifically, in addition to the NIDS mitigation, software/configuration mitigations such as Filter Network Traffic (M1037) and SSL/TLS Inspection (M1020) are also applied to mitigate the threat. In particular, SSL/TLS Inspection (M1020) is the mitigation that explicitly handles encrypted traffic; for such a configuration mitigation, the configuration settings check examines the component TC, for which it is assumed that the mitigation M1020 is correctly implemented.

For the refined Scenario #2 part in Fig. 12, SecureWeaver mitigates the threat T1040 on the remote access connection between the user and the thin client system. The secure protocol check function is used in this case to check 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 check inspects the remote desktop connection security via the IPSEC connection from the Usr to the VPN server (VPN) that is connected to RTR.

The refined Scenario #3 part in Fig. 12 includes the private cloud storage system and its supporting requirement (BKSys), and they are connected to the dmzLAN layer 3 network switch, which is of the L3SW component type. 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 check (M1048), a virtual machine is required as a host, which is the VM in Fig. 12. Moreover, the firewall (FW) that is placed between WAN and RTR satisfies the firewall use check (M1050). The network segmentation check is also successful as there are two LAN networks (bizLAN and dmzLAN) in Fig. 12, 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 validated via the configuration settings check.

We conclude that all the security threats in Fig. 12 are fully mitigated according to the security check conducted automatically by means of the seven check functions in SecureWeaver.

6.2. *Performance Evaluation*

In this subsection we evaluate the performance characteristics of SecureWeaver, and the overhead of the security check mechanism introduced in this paper. The first type of performance evaluation subjects SecureWeaver to increasingly complex input service requirements. The evaluation is conducted using the five scenarios introduced in Section 6.1.1. 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 check) needed to return a concrete and secure system design.

For the second type of performance evaluation we compare the results obtained with SecureWeaver when varying the number of security threats and the required security level for an in-depth analysis of complex system design feasibility and scalability. Five sets of experiments are performed for each scenario to obtain the numerical results presented in the following subsections.

6.2.1. *security check Performance*

The performance evaluation results of Scenario #1 to #5 are presented in Table 3, 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 check 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 check, $RSD_{t_{sec}}$ 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 security check mode).

Beside the performance evaluation results in Table 3, we include supplementary data in Table 4 with statistics on the number of components and relationships, and the temporary database used by SecureWeaver to store data during the refinement and security check 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 3, 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 4; 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. An difference of 30.6% is observed between Scenario #1 and #2 for n_{topo} in Table 3, 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 check. 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 check process 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 check mechanism only introduces a small overhead; for example, t_{sec}/t_{total} for the full scenario in Fig. 12 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.

For large and complex fully-abstract service requirements, SecureWeaver is technically limited to the available storage space for storing temporary data (applied actions and computed topology states). As shown in Table 4, $size_{action}$ and $size_{state}$ increase with the complexity of the input service requirement, and in Scenario #5 the size of temporary data reached a total of 28.4 GiB when using security (note that the available storage on the experiment machine was 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

Table 3: SecureWeaver evaluation results: security check 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	0.00032

Table 4: 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

#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 4.

6.2.2. *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 for all the included threats: 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. 13, 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 in general increases as the required security level is 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

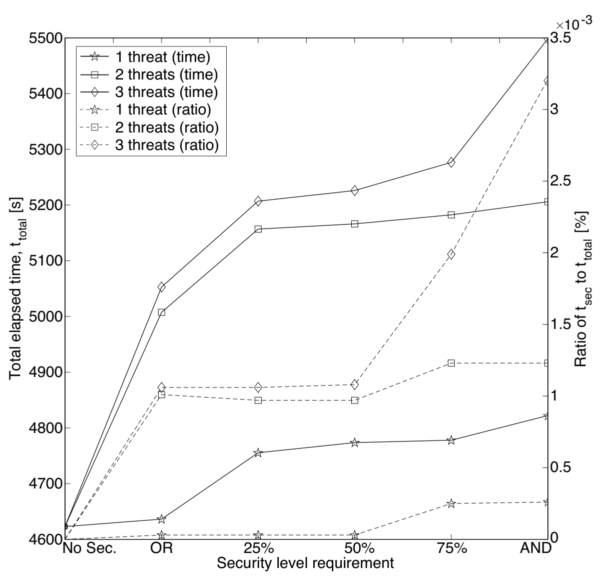


Figure 13: Effect of the security level requirement setting on SecureWeaver performance.

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 check 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 check. 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 (data not shown in the paper) 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. 13 as the security level requirements 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.

6.3. Feature Comparison

In this subsection we compare SecureWeaver 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. We also look into the method that is used to design or create the output, 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 shown in Table 5. For framework capabilities, both SecureWeaver and (Gressl et al., 2021) are able to perform both system design from abstract input and verify the security of the output, while (Scheid et al., 2017) and (Amato et al., 2018) are only capable of verifying a concretized input. As for (Kim et al., 2020), the framework is only capable of designing the security policy from an abstract input.

For the design method, both SecureWeaver and (Gressl et al., 2021) use DSE refine a concrete system, while (Scheid et al., 2017) uses clustering and external tools to manage its dependency process. The framework in (Amato et al., 2018) uses template matching to create the flow chain for verification, and (Kim et al., 2020) 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 (Ooi et al., 2022). 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 (Kim et al., 2020). For the quantitative-based approach, a numerical result is typically used to satisfy the quantitative security requirements, while a qualitative-based approach such as (Kim et al., 2020) 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 (Scheid et al., 2017) and (Amato et al., 2018) are based on abstract numerical values, (Gressl et al., 2021) 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.

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.

6.4. Discussion

Automated versus human design. The security check 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

Table 5: 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 (Scheid et al., 2017)	Verification	Clustering	NW	Quantitative (done via security score computation)	Virtual Network Function (VNF) & security score database
(Amato et al., 2018)	Verification	Template	IT/NW	Quantitative (done via security level assessment)	Pattern & security level database
IBCS (Kim et al., 2020)	Design	DFA & CFG	NW	Qualitative (done via policy generation)	Network Security Function (NSF) database
(Gressl et al., 2021)	Design & verification	DSE	IoT	Quantitative (done via probabilistic attack chain)	STRIDE based attack chain, security function database

be improved on by extending the set of refinement rules and the knowledge base).

Beyond the network domain. 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.

Internet versus intranet. A corporate network typically includes public web service accessible through the Internet, and internal services accessible from within the intranet. For example, an HTTPS gateway can be placed between a user and a web application, where an HTTPS tunnel between the user and the web application is terminated at the gateway, and the web application traffic between the gateway may be unencrypted within the organization intranet (e.g., HTTP). Such refinement rules can be designed by network and security experts and added into the existing set of refinement rules in SecureWeaver. While the current refinement rules in SecureWeaver do distinguish between intranet and the Internet (e.g., wire:LAN versus wire:extWAN), SecureWeaver itself does not distinguish between them from a security perspective, since we consider that the only way to have a fully-secure system is to have the traffic protected at all points, but such a distinction could be implemented as a system optimization mechanism. Do note, however, that there is no explicit distinction between intranet and Internet in the MITRE ATT&CK Enterprise matrix network domain, hence such a mechanism would need to be added based on another knowledge base.

Threat tagging and consistency. For scenarios that involve tagging related components and relationships with threats (e.g., threats involving the compromise of private data, such as T1040 for relationships and T1602 for components), SecureWeaver does not enforce any limitations nor perform consistency checks. Consequently, if there are any relevant MITRE ATT&CK threats both for components and relationships, they can be used to tag those components. Currently, if a user incorrectly tagged related entities with incompatible threats, SecureWeaver may not be able to find a valid solution, and the user needs to revise the input file from the point of view of threat placement. The solution that we are considering for this issue is to automatically place the threat(s) based on security best practices, a mechanism that we are planning to implement in the future.

Handling user devices. There may be scenarios in which the security of a user device can be compromised (in our scenario the user device is represented by the concrete component `Usr:User` shown in Fig. 11). In this paper we have assumed that the user device (as a concrete component) is not managed by the company, so SecureWeaver has no control over that user device, an illustration of the Bring-Your-Own-Device (BYOD) policy in a typical organization. However, if the user device is to be managed by the company, it should be added into the SecureWeaver service requirement as an abstract component, and in that case, rules can be used for SecureWeaver to refine it as part of the overall system design and ensure its security.

Zero trust. 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.

Feature comparison. Like any system, SecureWeaver has its advantages and disadvantages, as summarised in Table 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 check 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 check 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 threat(s). Currently, when an application expert considers a component/relationship to be susceptible to a certain security threat, that component/relationship needs be tagged explicitly with the corresponding threat id in the MITRE ATT&CK Enterprise matrix (network domain), and no higher-level labelling features exist (such as indicating that a component/relationship is involved with the storage or communication of sensitive information).

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.

7. Conclusion

In this paper we presented an intent-based secure system designer, named SecureWeaver, that based on a network service requirements input, which includes security requirements, is able to generate a system design that meets the specified functional, quantitative and security service requirements. SecureWeaver was implemented by leveraging the functionality of an existing intent-based system designer that targeted IT/NW services, named Weaver.

The core elements of the methodology that handle secure design aspects are the following. A security knowledge base that was implemented based on the MITRE ATT&CK framework makes it possible to establish the bidirectional relationship between security threats (attack techniques) and mitigation techniques in a comprehensive manner. Furthermore, a set of security check functions that are also implemented following the information provided in the MITRE ATT&CK framework make it possibly to automatically verify whether a certain system design mitigates the associated security threats or not.

The feasibility of our approach was illustrated via a typical corporate network scenario that included thin client systems, web systems and remote user access. Thus, 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.

We also evaluated the performance characteristics of the implementation, and demonstrated that the security check 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

Table 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 check mechanism, which incorporates best practices for mitigating attacks against specific system components
	A smaller number of independent security check 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 check 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

threats the overhead decreases sharply, reaching levels as low as 0.30% in our experiments. The paper also included a feature comparison with respect to research other works, emphasising the overall advantages of SecureWeaver.

As future work, we are considering to improve the usability of SecureWeaver by introducing a more generic method of labelling the entities with security-related information, such as "use of private data," which can then be employed by the system to automatically place the relevant MITRE ATT&CK threat(s) in a consistent manner, based on security best practices. We are also considering possible ways to optimize software performance by early elimination of those system designs that cannot be made secure by any means. Moreover, we consider extending the security knowledge base and security check functions to cover other areas than the network domain of the MITRE ATT&CK framework that we have addressed so far.

References

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

Barnum, S., 2012. Standardizing cyber threat intelligence information with the structured threat information expression (STIX). Mitre Corporation 11, 1–22.

Davoli, G., Cerroni, W., Tomovic, S., Buratti, C., Contoli, C., Callegati, F., 2019. Intent-based service management for heterogeneous software-defined infrastructure domains. International Journal of Network Management 29, e2051.

DesLauriers, J., Kiss, T., Pierantoni, G., Gesmier, G., Terstyanszky, G., 2021. Enabling modular design of an application-level autoscaling and orchestration framework using TOSCA-based application description templates, in: 11th International Workshop on Science Gateways, IWSG 2019, CEUR Workshop Proceedings. pp. 1–7.

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

Gressl, L., Steger, C., Neffe, U., 2021. Design space exploration for secure IoT devices and cyber-physical systems. ACM Transactions on Embedded Computing Systems (TECS) 20, 1–24.

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

Hernan, S., Lambert, S., Ostwald, T., Shostack, A., 2006. Uncover security design flaws using the STRIDE approach.

Jacobs, A.S., Pfitscher, R.J., Ribeiro, R.H., Ferreira, R.A., Granville, L.Z., Willinger, W., Rao, S.G., 2021. Hey, Lumi! Using natural language for intent-based network management, in: 2021 USENIX Annual Technical Conference (USENIX ATC 21), pp. 625–639.

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

Kim, J., Kim, E., Yang, J., Jeong, J., Kim, H., Hyun, S., Yang, H., Oh, J., Kim, Y., Hares, S., et al., 2020. IBCS: Intent-based cloud services for security applications. IEEE Communications Magazine 58, 45–51.

Køien, G.M., 2020. A philosophy of security architecture design. Wireless Personal Communications 113, 1615–1639.

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

Kuwahara, T., Kuroda, T., Osaki, T., Satoda, K., 2021. An intent-based system configuration design for IT/NW services with functional and quantitative constraints. IEICE Transactions on Communications E104.B, 791–804.

Kwon, R., Ashley, T., Castleberry, J., Mckenzie, P., Gourisetti, S.N.G., 2020. Cyber threat dictionary using MITRE ATT&CK matrix and NIST cybersecurity framework mapping, in: 2020 Resilience Week (RWS), IEEE. pp. 106–112.

Martin, L., 2014. Cyber kill chain. URL: `http://cyber.lockheedmartin.com/hubfs/GainingtheAdvantageCyberKillChain.pdf`.

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

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

Pham, M., Hoang, D.B., 2016. SDN applications–The intent-based northbound interface realisation for extended applications, in: 2016 IEEE NetSoft Conference and Workshops (NetSoft), IEEE. pp. 372–377.

Pimentel, A.D., 2020. A case for security-aware design-space exploration of embedded systems. Journal of Low Power Electronics and Applications 10, 22.

Rafiq, A., Mehmood, A., Ahmed Khan, T., Abbas, K., Afaq, M., Song, W.C., 2020. Intent-based end-to-end network service orchestration system for multi-platforms. Sustainability 12, 2782.

Rutkowski, M., Chris Lauwers, C., Curescu, C., 2020. TOSCA simple profile in YAML version 1.3. URL: `https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/TOSCA-Simple-Profile-YAML-v1.3.pdf`.

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

Strom, B., Applebaum, A., Miller, D., Nickels, K., Pennington, A., Thomas, C., 2018. MITRE ATT&CK: Design and Philosophy. The Mitre Corporation, McLean. Technical Report. VA, Technical report.

Wei, Y., Peng, M., Liu, Y., 2020. Intent-based networks for 6g: Insights and challenges. Digital Communications and Networks 6, 270–280.

Wu, C., Horiuchi, S., Murase, K., Kikushima, H., Tayama, K., 2021. Intent-driven cloud resource design framework to meet cloud performance requirements and its application to a cloud-sensor system. Journal of Cloud Computing 10, 1–22.