SecureWeaver: Intent-Driven Secure System Designer

Sian En Ooi
Razvan Beuran
Yasuo Tan
sianen.ooi@jaist.ac.jp
Japan Advanced Institute of Science and Technology
Nomi, Ishikawa, Japan

Takayuki Kuroda
Takuya Kuwahara
Norihito Fujita
NEC Corporation
Minato-ku, Tokyo, Japan

ABSTRACT

Design and management of networked systems, such as Information Technology/Network (IT/NW) or IoT systems, are inherently complex. Moreover, the need to adhere to security requirements adds even more complexity, as the manual audit and security mitigation of system design are time, skill, and labour intensive.

In this paper, we present SecureWeaver, a secure system designer that generates a system design which meets functional, quantitative and security service requirements. SecureWeaver is based on the intent-based designer for IT/NW services named Weaver, and security support was implemented by improving the Weaver design stage via a threat mitigation knowledge base, specific refinement rules, and a security verification mechanism. A case study on video surveillance service requirements is used to illustrate the security threats and their mitigation during the automatic design process. Our results show that SecureWeaver is able to mitigate and verify the solutions from a security perspective without incurring a significant overhead: in our experiments, average overhead is 0.04% for systems with more than 100 elements. We also present a feature comparison with three other related systems that emphasizes the practical advantages of SecureWeaver.

CCS CONCEPTS

• **Security and privacy** → **Security requirements**; • **Networks** → *Network management*; • **Social and professional topics** → *Systems analysis and design.*

KEYWORDS

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

ACM Reference Format:
Sian En Ooi, Razvan Beuran, Yasuo Tan, Takayuki Kuroda, Takuya Kuwahara, and Norihito Fujita. 2022. SecureWeaver: Intent-Driven Secure System Designer. In *Proceedings of the 2022 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems (SaT-CPS '22), April 27, 2022, Baltimore, MD, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3510547.3517923

SaT-CPS '22, April 27, 2022, Baltimore, MD, USA.

ACM ISBN 978-1-4503-9229-7/22/04…$15.00
https://doi.org/10.1145/3510547.3517923

1 INTRODUCTION

Deploying new services and managing information technology/ network (IT/NW) infrastructures are complex tasks, and the concepts of Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) have been introduced as a way to address these issues. However, there is a "communication" gap between service providers and enterprise customers, where the customers convey their needs in service-level requirement descriptions, whereas service providers primarily deal with resource-level requirements [20]. Translation between these types of requirements requires expert knowledge, as the service provider has to consider and prepare the IT/NW components and integrate them appropriately to meet customer's requirements. Approaches such as intent-based [6, 9, 14, 16, 20], template-based [2, 3, 13] solutions have been developed to address the issue.

Nevertheless, a system designer must not only meet the qualitative and quantitative requirements, but also verify whether the resulting system design is secure. The "secure by design" concept refers to a system that has been designed to be fundamentally secure. Some works, such as [8, 15], explore the idea of secure Design Space Exploration (DSE) as an integral part of the secure development process. This helps streamline the development and verify any potential security issues in the design stages, whereas a conventional non-automated design approach that involves a complete system design would require manual security audits and mitigation.

Similarly, the phenomenal growth of the Internet of Things (IoT) engenders the need for the fast and efficient deployment of IoT systems and cloud services. Since both the IT/NW and IoT domains involve networked systems that are made of heterogeneous components, we envisioned a secure system designer that is able to automatically design networked systems that are secure, regardless of their specific application domains.

While there are other automatic system design approaches [14, 16, 20] that can be used to design a system, they are usually limited to their specific domain, with selected cases for either architecture or parameter level design. Components, relationships and constraints in a topology graph are specified in architecture level design; in parameter level design, the parameters and fine tuning are specified according to the given topology. Works in parameter level design such as [20] have high flexibility compared to their architectural level design counterparts. However, these are incompatible with our use case, as we focus on architecture level design for IT/NW and IoT services.

Our work is based on Weaver [10, 11], an intent-based system designer for IT/NW services. Weaver builds on the DSE approach for designing a concrete system design by adding support for an

abstract service requirement as input, thus bridging the gap between providers and customers mentioned above. This makes it possible for Weaver to address architecture level design in a flexible and efficient manner. However, while Weaver can solve qualitative and quantitative constraints, it lacks a process to ensure that the designed system is secure.

We extended the existing Weaver to create SecureWeaver, a secure system designer that generates a system design which meets not only functionality requirements, but also security requirements. The core contributions of this paper are as follows:

- Proposed to employ the MITRE ATT&CK taxonomy to build the threat mitigation knowledge base used in SecureWeaver
- Extended Weaver to support logical and conceptual connections and added security-specific refinement rules
- Implemented a security verification mechanism to validate that all the threats present in the service requirement are mitigated in the generated system design

The remainder of the paper is organized as follows. Section 2 introduces research related to intent-based and secure system design. An overview of SecureWeaver is presented in Section 3. A brief description of Weaver is given in Section 4, followed by the SecureWeaver extension details in Section 5. The evaluation of SecureWeaver is presented in Section 6 from security verification, feature comparison and performance perspectives. The paper ends with a conclusion and references.

2 RELATED WORK

In this section, we discuss about the works related to SecureWeaver and their differences. The work in [9] is an intent-based security service automation for cloud services, IBCS (Intent-Based Cloud Services for Security Applications). It translates the user's network security intents into low-level security policies by extracting the data via Deterministic Finite Automata (DFA) method, maps the extracted data to the appropriate Network Security Function (NSF) as the low-level data, and generates the security policies using Context-Free Grammar (CFG) to be applied to the network interfaces. IBCS requires a ready cloud environment along with full knowledge of the target infrastructure security capabilities to be able to generate its security policy, whereas SecureWeaver designs from ground up a secure system that satisfies the given intent.

The work in [6] is an intent-based network management framework using natural language. It accepts natural language intent from an operator and extracts the required information using machine learning algorithm before passing the required information to an intent assembly module that generates concrete network configuration commands that meets the network management intent. This is an intent-based network management of existing network infrastructure, which does not meet our goal of automatically designing a secure system.

The work in [18] is an intent-based Network Function Virtualization (NFV) designer focusing on modelling and computing virtual network functions (VNF) chains to meet the quantitative and non-functional requirements such as security. In its quantitative computation, it requires definitions of abstract weights for the non-functional requirement, which may be hard to assign. These quantitative scores of VNF are clustered to their respective levels to find the VNF that satisfies the requirements. This is not as flexible as SecureWeaver in architecture design.

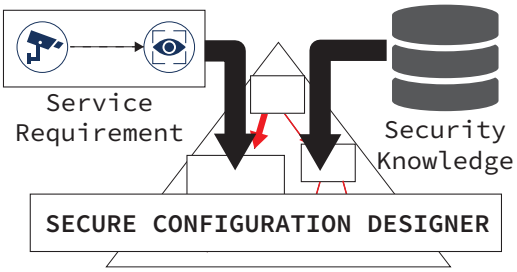

Figure 1: Overview of SecureWeaver.

The work in [1] is a model driven design for orchestrated cloud services which focuses on security level evaluation. It utilizes template-based matching to meet service requirement, where it also considers security propagation in the topology with numerical calculations to verify whether the template satisfies the security level. SecureWeaver, on the other hand, uses a known threat mitigation knowledge base which specifically maps the threat to their respective mitigation, and vice versa, which is more concrete than an abstract quantitative calculation.

The work in [4] is a DSE based designer for IoT and cyber-physical systems (CPS). It also uses the DSE approach to refine a system topology. The difference with SecureWeaver is that it is specifically used to design low-level IoT hardware and software, while SecureWeaver deals with high-level architecture design. Moreover, it uses Microsoft's STRIDE threat model to derive its attack types, while SecureWeaver utilizes MITRE ATT&CK taxonomy to define the threats and mitigations.

A more detailed comparison of some of these systems with SecureWeaver will be presented later in Section 6.2.

3 OVERVIEW OF SECUREWEAVER

Manual audit and vulnerability mitigations of a concrete system design in a production system is tedious in general. This typically requires a lot of expert human effort and takes relatively long time to resolve security issues. Fig. 1 illustrates the main motivation behind SecureWeaver, where we extend Weaver with security in mind to design secure system design. By integrating a security knowledge base into Weaver with relevant modifications, we envisioned that the system designer is able to consult the knowledge base regarding entities in its topology for security issues and mitigate them with appropriate mitigation techniques.

The concepts regarding the components and their relationship in Weaver have to be extended to create SecureWeaver. 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.

The next two sections will introduce those aspects of the original Weaver that are related to SecureWeaver (Section 4), and the implementation of security support in SecureWeaver (Section 5).

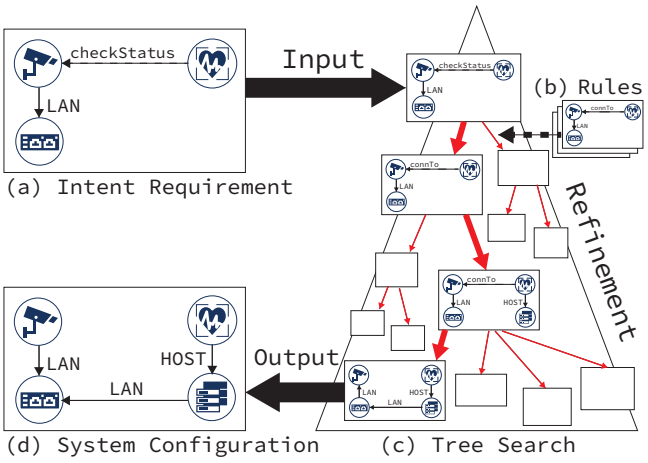

Figure 2: Weaver processing flow.

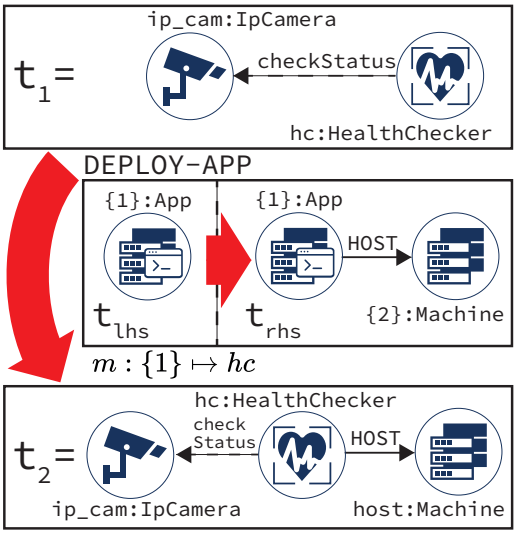

Figure 3: Example of a topology refinement step.

4 WEAVER SYSTEM DESIGNER

Our approach is based on the Weaver IT/NW system designer [10, 11]. Fig. 2 gives an overview of the Weaver processing flow, where the rectangular boxes denote the state of the topology at a particular time. The arrows in the pyramid are refinements done following the matching rules in Weaver.

A top-level description of the intended outcome (intent) is the service requirement input, inspired by the concept of intent-based networking (IBN). The abstract entities in the input are then continually refined using tree search with matching refinement rules until the final topology is completely void of abstract entities. The final topology, which is said to be concrete is then output by Weaver as a system design for the actual implementation of the intent. The following subsections will briefly describe the Weaver data format, rules and topology refinement, tree search, and system design output.

4.1 Data Format Definitions

The data format used in Weaver is structured similarly to TOSCA (Topology and Orchestration Specification for Cloud Applications) [17], a specification that declaratively describe service configuration in a topology to facilitate provisioning automation via definition of components, their relationships, and their individual attributes.

A component is a pair "$v : ctype$", where v is the component identifier and $ctype$ is its type. A component type is denoted 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. A generic operating system (OS) is an abstract component , whereas "Ubuntu" is a concrete component, which can be derived from the abstract OS component via inheritance.

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

4.2 Rules and Topology Refinement

In order to refine an abstract service requirement input into a completely concrete topology, refinement process is repetitively done to transform a topology from one state to another using matched refinement rule. A refinement rule, r, is a one-step refinement process that is denoted as a tuple $r = (t_{lhs}, t_{rhs})$, where t_{lhs} is the left-hand side and t_{rhs} is the right-hand side of a rule. An example of r is illustrated in Fig. 3 as "DEPLOY-APP". 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. In short, the topology refinement process can be illustrated as Fig. 3, where the health checker application, hc in topology t_1 is transformed into t_2 by rule "DEPLOY-APP" with match, $m : \{1\} \mapsto hc$. 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 *HealthChecker* component type. Hence, component $\{1\}$ of type *App* is equivalent to hc of type *HealthChecker*, such that $\{1\}$ in the rule "DEPLOY-APP" can be replaced by $\{1\} \underset{HOST}{\longrightarrow} \{2\}$.

4.3 Tree Search-based Algorithm for DSE

Weaver finds refinement candidates through a deterministic search process, where it iteratively apply relevant actions to obtain a completely concrete topology. The search algorithm exits when all entities in the topology are fully concretized; detailed information on the tree search algorithm and topology concreteness definition can be found in [11].

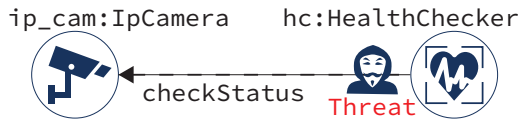

Figure 4: Service requirement with threat.

5 SECURITY SUPPORT ENHANCEMENT

In this section we discuss the extension of Weaver to: (i) accommodate threat information in the topology; (ii) formulate rules that are compatible with our threat mitigation approach; (iii) discover threats and verify their mitigation in the generated topology.

5.1 Security Threats

The blue Anonymous mask icon in Fig. 4 represents an explicitly defined threat in the relationship. We define a threat as the path of an attack to a target that requires protection. Fig. 4 shows a threat in the relationship between an IP camera, *ip_cam* and a health checker application, *hc*, where *checkStatus* is susceptible to threats such as sniffing or eavesdropping.

In order to model threats and their inheritance for the affected branches in a topology, the root relationship between two component (e.g., *checkStatus* between *ip_cam* and *hc*) has to be preserved in the topology to verify its threat and possible mitigations. Hence, a new approach to how Weaver transform its topology is required to accommodate the threat mitigation information. For this paper, we only consider threats in the relationship between two components, and a threat needs to be explicitly defined in the service configuration for it to be taken into account.

5.2 Threat Mitigation Knowledge Base

There are various methodologies for security assessment, such as Lockheed Martin Cyber Kill Chain (CKC) [12], MITRE ATT&CK [19], and Microsoft STRIDE [5]. The CKC is a well known intrusion-centric framework that describes a well defined sequence of attack. ATT&CK on the other hand is more like a list of techniques by tactics, where it does not propose any specific order of operation. STRIDE is a high-level threat model, commonly used during security development life-cycle as its focused on identifying overall categories of attacks.

Since a threat is required to be explicitly defined 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 have sufficient security context. For our threat mitigation knowledge base, we chose ATT&CK as it provides a rich taxonomy of adversarial tactics, techniques, and common knowledge that can be readily used in various scenarios.

5.2.1 MITRE ATT&CK. The ATT&CK taxonomy is categorised into tactics, techniques, sub-techniques, and mitigations. There are a total number of 14 tactics, 185 techniques, 367 sub-techniques and 42 mitigations in ATT&CK. To create the knowledge base, we are mainly interested in threats and mitigations. Therefore, we designate ATT&CK techniques and mitigations as the threats and mitigations in the knowledge base.

The total number of techniques and sub-techniques in ATT&CK is large. Hence, we started filtering the possible ATT&CK mitigations that are directly relevant to Weaver. This means that the selected ATT&CK mitigations can be directly mapped to Weaver existing *rtype* structure without any modifications. We will be using the ATT&CK Network Sniffing (T1040) technique and Encrypt Sensitive Information (M1041) mitigation for all the threat and mitigation examples in this paper.

5.2.2 Threat Mitigation Data Structure. For the data structure in the threat mitigation knowledge base, we implemented a JSON key-value pair representation. The knowledge base keys are secure protocols, and the values are the corresponding ATT&CK mitigation and threat identifiers as a string. Some examples of the key-value pairs used in this paper are: "*IPSEC*" : "*M*1041.*T*1040", "*SRTP*" : "*M*1041.*T*1040", and "*HTTPS*" : "*M*1041.*T*1040". This means that when an IPSEC *rtype* appears as the relationship between two components, the mitigation M1041 is used to counter threat T1040.

5.3 Data Representation

In order to add security support, we have to be able to explicitly define a threat in the service requirement and modify Weaver's refinement process to identify and verify security threats and mitigations in the topologies. For the first step, we utilize Weaver relationship properties data format to store the threat information (e.g., "T1040") in the service requirement. The remaining steps require an extension of Weaver that was done by preserving Weaver's architecture and reusing its existing functionality as much as possible, as explained next.

5.3.1 Logical and Conceptual Connections. In Section 5.1 we described the issue with the rewriting and removal of relationships between the root components. Security requires more context from the topologies than just physical connections, and for this purpose we introduce two new types of connections: logical and conceptual.

We define a logical connection as a relationship with an *rtype* that corresponds to the TCP/IP model's application and network layer protocols. The TCP/IP model is well suited for describing the relationship between two components, and we determined that considering only the TCP/IP model's application and network layers is currently sufficient. 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, *HTTP* and *IPSEC*, which are illustrated by dash-dotted lines.

A conceptual connection, *CC*, 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 correspond logically to a service specified in that requirement. Fig. 5 includes two groups, one made of the *IpCamera*, which forms a group by itself, and another made of the *HealthChecker* and the *HOST*, as the latter was added to the topology via a refinement rule corresponding to the *HealthChecker*. The conceptual connection between these two groups is the pair of *HTTP* and *IPSEC* logical connections. Retaining such logical connections in the topology makes it

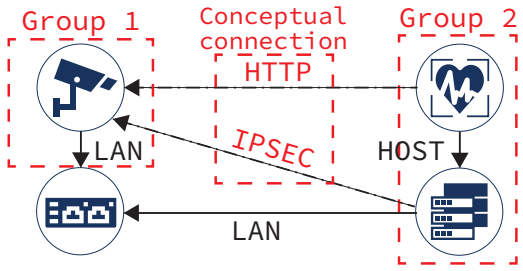

Figure 5: Conceptual connection between 2 component groups.

possible to preserve information needed by the subsequent security verification algorithm.

5.3.2 Refinement Rules. The refinement rules in Weaver transform one topology state to another by removing or changing abstract *ctype* or *rtype* entities to achieve a concrete state. Even for the concrete logical connections in SecureWeaver, the refinement rules have to be designed in a way that ensures the logical connections are preserved in the topology.

When it comes to security concerns in refinement rules, we have only explicitly included them in the service requirement. The threat and mitigation are implicitly included through the TCP/IP application and network layer protocols' inherent security properties when used as an *rtype* in a topology, and they are defined in the threat mitigation knowledge base for security verification purposes.

During topology refinement, there may be more than one solution to mitigate a threat. Fig. 6 illustrates a service requirement, t_0 with *HealthChecker* and *IpCamera* connected by *checkStatus* with threat T1040. The *checkStatus* can be either achieved using *HTTP* or *HTTPS* application protocol, where *HTTP* is insecure and *HTTPS* is secure. The left and right topology in Fig. 6 show the sequential topology refinements for *HTTP* and *HTTPS*, respectively.

In the left topology state, *HTTP* does not mitigates the threat T1040, for which an implicit threat is illustrated on the affected relationship, denoted as a white Anonymous icon. Following the refinement procedure of the left topology, the threat is implicitly inherited by the *connTo* abstract *rtype* as shown in $t_{i+n+1,1}$. 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. In topology state $t_{i+n+2,1}$, the abstract *connTo rtype* is replaced with a concrete *IPSEC* logical connection. At this point, the left topology's threat is mitigated as IPSEC is defined as a mitigation for T1040 in the threat mitigation knowledge base. A mitigation is illustrated as a blue Anonymous mask with a red "X" icon as shown in Fig. 6.

For the right topology state, *HTTPS* mitigates the T1040 as defined in the threat mitigation knowledge base. Hence, the remaining refinements to concretize the right topology do not have any security issue. This shows one should not eliminate logical connections that do not mitigate a threat at first discovery by Weaver, as it would truncate other possible solutions. Therefore, for the security aware Weaver, we only perform security verification after all the entities in the topology are concretized.

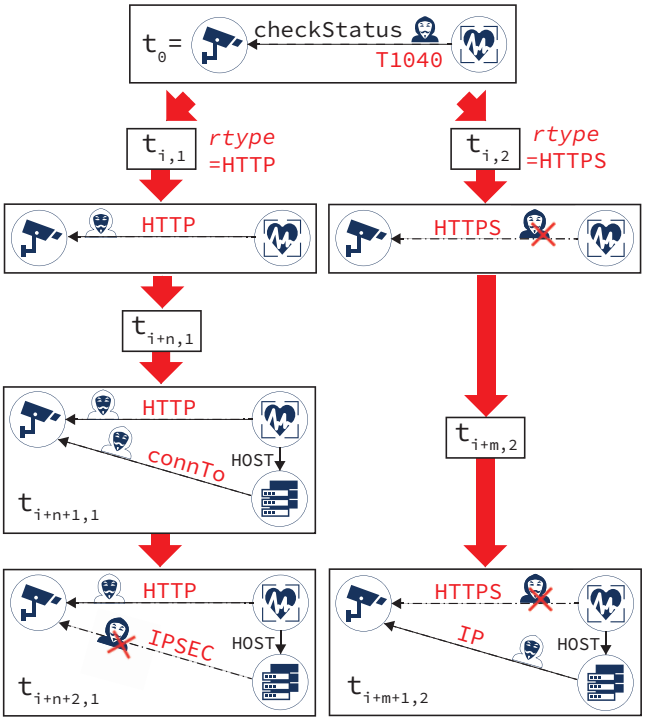

Figure 6: Example of possible refinements that mitigate the threat defined in service requirement.

Table 1: Example of security verification outcome for a given conceptual connection.

Threat in security requirement	Applied mitigation	Secure?
N/A	N/A	Yes
N/A	M1041.T1040	Yes
T1040	M1041.T1040	Yes
T1176	M1041.T1040	No
T1040	N/A	No

5.4 Security Verification Mechanism

In order to verify whether a threat in a topology is mitigated or not, we implemented a security verification algorithm that ensures every threat specified in the service requirement is addressed before Weaver outputs the system design. In general, we only want to mitigate the threats that are explicitly defined in the service requirement, where a system design is secure if all the threats defined in the service requirement are mitigated. Table 1 shows the expected security verification results for each conceptual connection. For the case where the security requirement is Browser Extensions, T1176, if the mitigation applied in the conceptual connection does not match its defined threat, the resultant system design is also considered as insecure.

The security verification is done after Weaver verifies that both topology and aspects have no abstract entities. Hence, the entry

point for the security verification algorithm is declared in the topology objective verification function in Weaver. We divided the security verification process into three main parts.

5.4.1 Retrieve Connection Threats from Service Requirement. The first part of the security verification process (topology.is_secure()) searches for all the relationship threats e_{threat} in the service requirement input, t_0, and store them into an array. A CC_{threat} is an relationship in the service requirement (top-level) that has a threat defined in the relationship property. The "topology" prefix of the $is\_secure()$ function denotes that the function is a class method of a topology class, where a topology class holds a single topology state in Weaver. After retrieving all the e_{threat}, the algorithm verify each of the CC_{threat} corresponding to e_{threat} at the second part of the security verification process. When an unmitigated CC_{threat} is determined, the process will return False to signify that the target topology is insecure. In this case where one of the objective fails, the selected topology state will be discarded and a new topology state will be refined and the verification process is repeated.

Algorithm 1 Search and verify the mitigation of threats in conceptual connections

Input: Edge src., e_{src}, Edge dst., e_{dst}, Edge threat, e_{threat}
Output: True or False

```
1: function TOPOLOGY.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 ! = "wire:lan" then
5:             if e_conn,type prefix ! = "wire:" then
6:                 if e_conn,dst == e_dst then
7:                     CC_mitigation ← e_conn mitigation info
8:                     if CC_mitigation,threat == e_threat and
   CC_mitigation ! = ∅ then
9:                         return True
10:                    end if
11:                else
12:                    e_dst,child ← e_dst child that is connected to
                       e_conn,dst
13:                    if e_dst,child ! = ∅ then
14:                        CC_mitigation ← e_conn mitigation info
15:                        if CC_mitigation,threat == e_threat and
   CC_mitigation ! = ∅ then
16:                            return True
17:                        end if
18:                    end if
19:                end if
20:            else
21:                if topology.verify_CC(e_conn,dst, e_dst, e_threat)
   then
22:                    return True
23:                end if
24:            end if
25:        end if
26:    end for
27:    return False
28: end function
```

5.4.2 Conceptual Connection Security Verification. The second part of the security verification process (topology.verify_CC()) verifies the CC_{threat} that is passed on by the first part of the verification process. Algorithm 1 takes CC_{threat} as an input and verify whether the input's threat is mitigated according to Table 1. To do this, all connections (relationships) from the input relationship's source component to other components are first obtained and stored in an array $e_{conn}[]$ as shown in line 2 in Algorithm 1. Each e_{conn} in the array are checked whether they are related to CC_{threat}. An application layer protocol relationship in a e_{conn} can be determined via comparing the e_{conn} destination to the CC_{threat} destination as shown in line 6 in Algorithm 1. If one of the logical connection of a conceptual connection is found, the $rtype$ information is then retrieved from the threat mitigation knowledge base as shown in line 7 in Algorithm 1.

If the e_{conn} does not have the prefix "wire:lan" and it fails the first check, this means the relationship is a network layer of the TCP/IP model, where CC_{threat} source component may be connected to a child of the CC_{threat} destination component. For this case, we have information about the relationship $rtype$, relationship root source and destination components. Hence, we have to determine whether the destination component of e_{conn} is a child of CC_{threat} destination component. The function "topology.verify_node_child()" is checked recursively whether the target component is a child of a root component; it takes a target and root components as input, and outputs True or False as result.

Up to this point, the assumption is that both application and network layer relationships are connected from a root source component to their respective destination components. For cases where the network layer relationship does not connect from a root source component, line 21 in Algorithm 1 calls recursively the topology.is_secure() function with the e_{conn} relationship's destination component, replacing the CC_{threat} root source component while the other inputs stay constant. By doing so, Algorithm 1 is able to flexibly verify topologies that have their application and network layer relationships connected to components in any orientations.

5.4.3 Threat Mitigation Retrieval from Knowledge Base. The third part of the security verification process is the function named edge.retrieve_mitigation(), which retrieves the threat mitigation information of an application or network layer relationship from the knowledge base. In Algorithm 1, the threat mitigation information is retrieved when a relationship is verified to be of the target conceptual connection. As noted in the prefix of the threat mitigation retrieval function, the function is defined as a class of a relationship (edge), where it could be efficiently referenced.

Using the *name* of a relationship $rtype$, the retrieval function search through the threat mitigation knowledge base for the matching *name* key. This JSON key-value structure is sufficient for simple evaluations of such approach. However, since a threat and/or mitigation may also reference to multiple application, network protocols and many more entities, a better way to structure the knowledge base would be by structuring the threat mitigation information in an ontology. One example of ATT&CK based ontology is [7], where the threat and mitigation (offensive technique) taxonomy is mapped with the MITRE D3FEND (defensive technique) knowledge

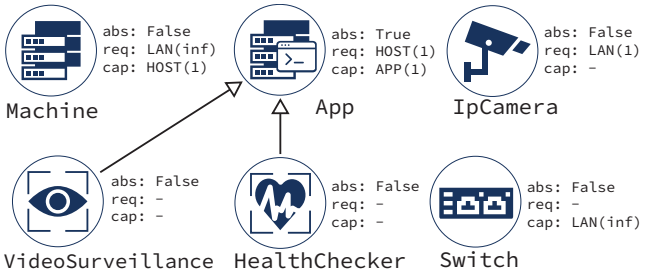

Figure 7: Properties and relationships of components.

via digital artifacts. We plan to implement an ontology-based threat mitigation knowledge base as a future work.

6 EVALUATION

In this section, we evaluate the proposed SecureWeaver system described in Section 5. First, SecureWeaver is evaluated functionally: we utilize SecureWeaver to generate a secure system design given a service requirement that includes threats, and verify that they are mitigated in all the possible concrete topologies. Then, we compare SecureWeaver to other related systems to highlight their pros and cons. Lastly, we conducted a performance evaluation on SecureWeaver to determine its security verification overhead.

6.1 Functionality Evaluation

SecureWeaver is implemented in Python 3, and the experiments we present here were performed with an Amazon Web Services (AWS) Elastic Compute Cloud (EC2) VM instance. The EC2 instance utilized is the "t2.micro", which features one virtual CPU with a frequency of up to 3.3 GHz and 1 GB of RAM.

6.1.1 Service Requirement for Evaluation. For this evaluation, we use the following components to build the service requirement, where all the component types and their deriving relations are shown in Fig. 7. Both video surveillance and health checker component types are derived from *App* component type while the component type's abstractness, *abs*, capability, *cap* and requirement, *req* are also shown in Fig. 7. An *IpCamera* has $req = LAN(1)$ while a *Switch* type has $cap = LAN(inf)$, where *LAN* is an *rtype* and the number in the *rtype* corresponds to the minimum or maximum number that the same *rtype* can be "connected" to. This means that when an *IpCamera* is defined in a service requirement, the final topology must have a maximum number of one *Switch* "connected" to it, while the *Switch* can be "connected" to an infinite number of component type with $req = LAN$.

In this evaluation, we assume a customer has an IP camera in the office and wants to sign up for a comprehensive security package that includes video surveillance and health checking services. The corresponding service requirement shown in Fig. 8, where each icon represents a component type labeled with its *nid* and *ctype*. The component *ip_cam* of type *IpCamera* represents the existing IP camera in the customer's office, while the *hc* of type *HealthChecker* and *vs* of type *VideoSurveillance* are the services that are to-be deployed. Functional requirements of the new services are denoted as abstract relationships of type *checkStatus*, $e_{sendStatus}$, and *sendVideo*, $e_{sendVideo}$, where $e_{checkStatus}$ means *hc* has to regularly monitor *ip_cam* status and $e_{sendVideo}$ means *ip_cam* is required to stream its video feed to *vs*. The threats defined in *checkStatus* and *sendVideo* are both ATT&CK defined network sniffing, T1040, where there may be a threat to the confidentiality of the communication between the IP camera and its applications.

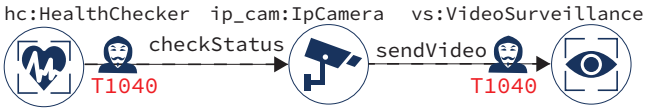

Figure 8: Evaluation service requirement.

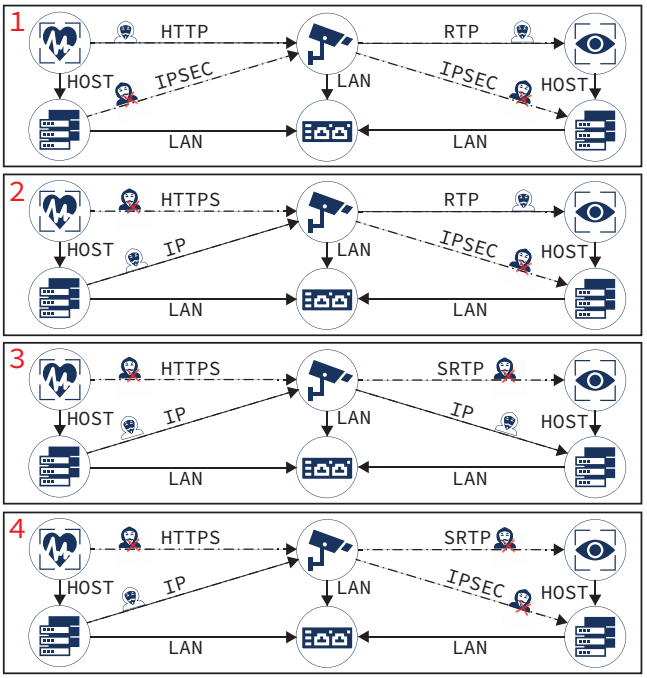

Figure 9: Examples of topologies for which the security verification was successful.

6.1.2 System Design Security Verification. After providing SecureWeaver with the evaluation service requirement, it generates a concrete and secure system design that corresponds to the input service requirement.

SecureWeaver has two mode of operation: automatic and interactive. In automatic mode, SecureWeaver will by default match and apply the first refinement rule that satisfies both quantitative and qualitative requirements, and only output the first system design that meets the quantitative, qualitative and security requirements. In interactive mode, SecureWeaver displays all valid refinement rules that can be applied to the current state of the topology, and the user can manually choose the next refinement rule to be applied.

Examples of possible concrete and secure SCs are shown in Fig. 9, where the topology with denoted with "1" in the top-left corner is the default system design output in automatic mode, and the other topologies are generated via interactive mode.

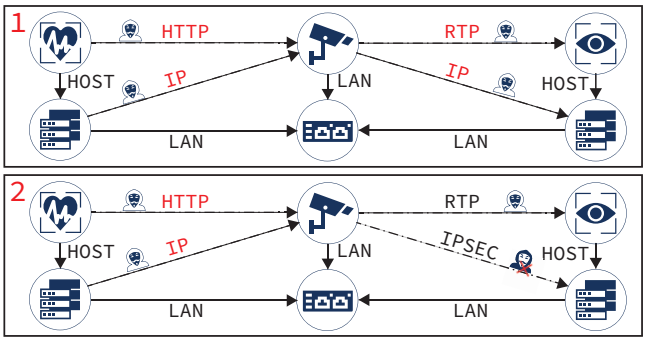

Figure 10: Examples of topologies rejected by the security verification algorithm.

For the first system design in Fig. 9, *hc* and *vs* are communicating with *ip_cam* via inherently insecure application layer protocols (*HTTP* and *RTP*), which do not mitigate the T1040 threats defined in the service requirement. Hence, SecureWeaver secures the topology by applied *IPSEC* network protocol for both conceptual connections to mitigate their respective threats (*IPSEC* mitigates T1040 as M1041, encrypt sensitive information). For system design 2, the communication between *hc* and *ip_cam* is replaced with *HTTPS* in the application layer, which addresses T1040 via M1041; hence, the plain *IP* network layer protocol can be used for the conceptual connection between the *hc* and *ip_cam*. For system design 3, both *hc* and *vs* are communicating via *HTTPS* and *SRTP* on the application layer, both protocols mitigating T1040 via M1041. It is also possible to "oversecure" a conceptual connection, as shown in system design 4: *SRTP* already mitigates the threat between *hc* and *ip_cam* at the application layer, but *IPSEC* is used at the network layer.

We also present two examples of rejected topologies that are concrete but insecure, as shown in Fig. 10. These topologies do not appear as the final system design as they are rejected during the refinement and verification process. For the first rejected topology, both conceptual connection between *hc*, *vs* and *ip_cam* are insecure as both application and network layer protocol does not mitigates the threat. In the second rejected topology, while the conceptual connection between *vs* and *ip_cam* is secure, the topology is rejected due to the insecure conceptual connection between *hc* and *ip_cam*.

6.2 Feature Comparison

In this subsection, we compare SecureWeaver to the related works discussed in Section 2. The comparison will mainly cover what each automatic system designer can do, especially on security. Table 2 shows the summary of the feature comparisons, where the method of system design, target domain usage, security threat and mitigation approach and security knowledge base is compared. For the system design method, both SecureWeaver and [4] use DSE, while [1] uses template matching and [18] uses clustering and external tools to manage its dependency process. In general, template-based approaches are more rigid than search-based design.

For the target domain for the system designer, SecureWeaver covers all three aspects where the others are specifically used for IT/NW or IoT. Furthermore, all three related works security threat mitigation approaches are quantitative-based, where numerical result is used to satisfy the quantitative security requirement. SecureWeaver mitigates threat in a qualitative manner, where we determine that the target threats are mitigated via threat mitigation from its knowledge base which is a matching problem and not as an optimization problem. In this case, as long as all service requirements are met, the designed system is satisfactory, which also results faster and efficient design process.

All three related works use databases that store the set/ template/attack chain and their numerical values for security calculations. While [18] and [1] are based on abstract numerical values, [4] designed their attack chain based on the STRIDE model. The SecureWeaver database is based on ATT&CK for a more concrete and comprehensive coverage. Thus, SecureWeaver is well suited for addressing architecture level design in IT/NW and IoT domains.

6.3 Performance Evaluation

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

6.3.1 Expected System Design Output. The service requirement used is similar to that in Fig. 8, where each office contains a single IP camera and it is connected to a health checker and video surveillance application. When increasing the number of offices, the number of IP cameras also increase linearly along with the video surveillance applications as a pair. For the health checker application, there is only one instance, and all IP camera will send their status to it. Hence, the expected system design output using SecureWeaver in automatic mode is shown in Fig. 11, where each office has an IP camera, a LAN switch and a video surveillance application, and the health checker application connects to every office's IP camera. Both *HTTP* and *IPSEC* are chosen by default in automatic mode as the application and network layer protocol in the their relationship, respectively.

6.3.2 Performance Evaluation Results. The performance evaluation results are shown in Table 3, where n is the number of offices, n_{comp} is the number of components in the system design, n_{rel} is the number of relationships in the system design, n_{all} is the total number of components and relationships in the system design, n_{threat} is the number of service requirement threats, n_{iter} is the number of algorithm iterations, n_{topo} is the number of topology concreteness and quantitative checks, n_{sec} is the number of security verifications,n_{child} is the number of component dependency searches, t_{total} is the total time elapsed in seconds to design and verify the system design, $RSD_{t_{total}}$ is the relative standard deviation of t_{total}, t_{sec} is the time elapsed in seconds for security verification,

Table 2: Feature comparison of SecureWeaver with related works.

Name	Method	Target	Threat Mitigation	Security Knowledge
SecureWeaver	DSE	IT/NW, IoT	Qualitative, via security verification	ATT&CK database
INSpIRE [18]	Clustering	IT/NW	Quantitative, via score calculation	VNF & score database
[1]	Template	IT/NW	Quantitative, via security level calculation	Pattern & security level database
[4]	DSE	IoT	Quantitative, via probabilistic attack chain calculation	STRIDE based attack chain, security function database

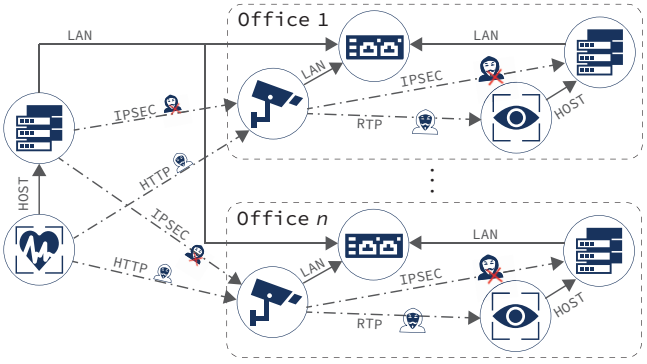

Figure 11: Expected output topology and security configuration.

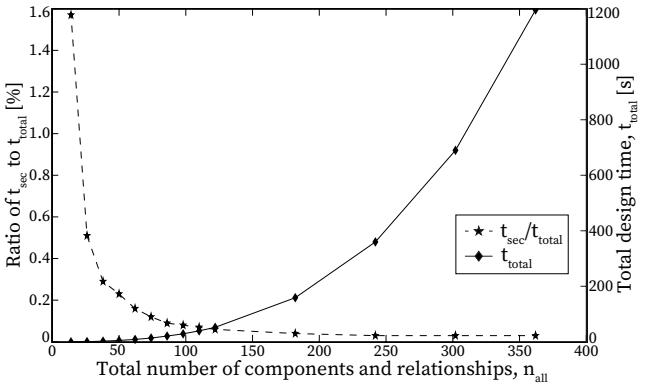

Figure 12: Performance evaluation results for SecureWeaver.

$RSD_{t_{sec}}$ is the relative standard deviation of t_{sec}, and t_{sec}/t_{total} is the percentage of t_{sec} from the total t_{total}.

The results in Table 3 are the averages of 10 experiments for each n number of offices. As both n_{comp} and n_{rel} increase, the average t_{total} increased in a manner that can be curved fitted ($R^2 = 1$) as shown in Equation 1, where n_{all} is the sum of n_{comp} and n_{rel}:

$$t_{total} = 3 \cdot 10^{-5} \cdot n_{all}^3 - 0.001 \cdot n_{all}^2 + 0.1834 \cdot n_{all} - 3.9823. \quad (1)$$

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

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

A more detailed analysis shows that the above performance is an intrinsic characteristic of the original Weaver (and we plan to use artificial intelligence to increase its efficiency). When considering only the security verification part of the system, we note that the average elapsed time, t_{sec}, increases much slower. Thus, when expressed in percentages, t_{sec} versus t_{total} decreases sharply from 1.57% for $n = 1$ to 0.03% for $n = 30$. These results are plotted and shown in Fig. 12. For real-world system designs have more than 100 entities (9 to 30 offices), the overheads are 0.07% at $n = 9$ to 0.03% at $n = 30$ with an average overhead of 0.04%. This means that our SecureWeaver extension only incurs a small overhead, which becomes comparably very small with respect to the total processing time as n_{comp} and n_{rel} increase.

7 CONCLUSION

In this paper, we presented SecureWeaver, a secure system designer based on Weaver that generates a system design that meets functional, quantitative and security service requirements. We discussed its usefulness in creating secure system design through a case study and feature comparison with related works, and showed its feasibility through performance evaluation.

We evaluated SecureWeaver using an IoT video surveillance service requirement case study, where the results showed that it is able to design secure system from an abstract service requirement that includes security threats. We also presented a feature comparison between SecureWeaver and three other related works that emphasize the advantages of SecureWeaver. Lastly, in the performance evaluation we demonstrated that the new security verification algorithm in SecureWeaver only incurs a small overhead with an average of 0.04% of the total design time for systems containing 100 or more entities, which underlines the feasibility of the proposed approach.

REFERENCES

[1] Flora Amato, Nicola Mazzocca, and Francesco Moscato. 2018. Model driven design and evaluation of security level in orchestrated cloud services. *Journal of Network and Computer Applications* 106 (2018), 78–89.

[2] James DesLauriers, Tamas Kiss, Gabriele Pierantoni, Gregoire Gesmier, and Gabor Terstyanszky. 2021. Enabling modular design of an application-level auto-scaling and orchestration framework using tosca-based application description templates.

Table 3: Numerical results for the performance evaluation of SecureWeaver.

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

In *11th International Workshop on Science Gateways, IWSG 2019*. CEUR Workshop Proceedings.

[3] Charafeddine El Houssaini, Mahmoud Nassar, and Abdelaziz Kriouile. 2015. A cloud service template for enabling accurate cloud adoption and migration. In *2015 International Conference on Cloud Technologies and Applications (CloudTech)*. IEEE, 1–6.

[4] Lukas Gressl, Christian Steger, and Ulrich Neffe. 2021. Design Space Exploration for Secure IoT Devices and Cyber-Physical Systems. *ACM Transactions on Embedded Computing Systems (TECS)* 20, 4 (2021), 1–24.

[5] S Hernan, S Lambert, T Ostwald, and A Shostack. 2006. Uncover Security Design Flaws Using The STRIDE Approach.

[6] Arthur S Jacobs, Ricardo J Pfitscher, Rafael H Ribeiro, Ronaldo A Ferreira, Lisandro Z Granville, Walter Willinger, and Sanjay G Rao. 2021. Hey, Lumi! Using Natural Language for {Intent-Based} Network Management. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 625–639.

[7] Peter E Kaloroumakis and Michael J Smith. 2021. *Toward a Knowledge Graph of Cybersecurity Countermeasures*. Technical Report. Technical report.

[8] Eunsuk Kang. 2016. Design space exploration for security. In *2016 IEEE Cybersecurity Development (SecDev)*. IEEE, 30–36.

[9] Jinyong Kim, Eunsoo Kim, Jinhyuk Yang, Jaehoon Jeong, Hyoungshick Kim, Sangwon Hyun, Hyunsik Yang, Jaewook Oh, Younghan Kim, Susan Hares, et al. 2020. IBCS: intent-based cloud Services for Security Applications. *IEEE Communications Magazine* 58, 4 (2020), 45–51.

[10] Takayuki Kuroda, Takuya Kuwahara, Takashi Maruyama, Kozo Satoda, Hideyuki Shimonishi, Takao Osaki, and Katsushi Matsuda. 2019. Weaver: A Novel Configuration Designer for IT/NW Services in Heterogeneous Environments. In *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–6.

[11] Takuya Kuwahara, Takayuki Kuroda, Takao Osaki, and Kozo Satoda. 2021. An intent-based system configuration design for IT/NW services with functional and quantitative constraints. *IEICE Transactions on Communications* E104.B, 7 (2021), 791–804.

[12] Lockheed Martin. 2014. Cyber kill chain. http://cyber.lockheedmartin.com/hubfs/GainingtheAdvantageCyberKillChain.pdf

[13] Nicolae Paladi, Antonis Michalas, and Hai-Van Dang. 2018. Towards secure cloud orchestration for multi-cloud deployments. In *Proceedings of the 5th Workshop on CrossCloud Infrastructures & Platforms*. 1–6.

[14] Minh Pham and Doan B Hoang. 2016. SDN applications-The intent-based Northbound Interface realisation for extended applications. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE, 372–377.

[15] Andy D Pimentel. 2020. A case for security-aware design-space exploration of embedded systems. *Journal of Low Power Electronics and Applications* 10, 3 (2020), 22.

[16] Adeel Rafiq, Asif Mehmood, Talha Ahmed Khan, Khizar Abbas, Muhammad Afaq, and Wang-Cheol Song. 2020. Intent-based end-to-end network service orchestration system for multi-platforms. *Sustainability* 12, 7 (2020), 2782.

[17] Matt Rutkowski, CN Chris Lauwers, and C Curescu. 2020. TOSCA Simple Profile in YAML Version 1.3. https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/TOSCA-Simple-Profile-YAML-v1.3.pdf

[18] Eder J Scheid, Cristian C Machado, Muriel F Franco, Ricardo L dos Santos, Ricardo P Pfitscher, Alberto E Schaeffer-Filho, and Lisandro Z Granville. 2017. INSpIRE: Integrated NFV-based intent refinement environment. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 186–194.

[19] BE Strom, A Applebaum, DP Miller, KC Nickels, AG Pennington, and CB Thomas. 2018. *MITRE ATT&CK: Design and Philosophy. The Mitre Corporation, McLean*. Technical Report. VA, Technical report.

[20] Chao Wu, Shingo Horiuchi, Kenji Murase, Hiroaki Kikushima, and Kenichi Tayama. 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 (2021), 1–22.