Game of Zones: An Automated Intent-Based Network Micro-segmentation Methodology

Daniele Canavese, Romain Laborde, Abir Laraba, Afonso Ferreira, Abdelmalek Benzekri

Game of Zones: An Automated Intent-Based Network Micro-segmentation Methodology

Daniele Canavese
IRIT, CNRS
Toulouse, France
daniele.canavese@irit.fr

Romain Laborde
IRIT, UT3
Toulouse, France
romain.laborde@irit.fr

Abir Laraba
IRIT, UT3
Toulouse, France
abir.laraba@irit.fr

Afonso Ferreira
IRIT, CNRS
Toulouse, France
afonso.ferreira@irit.fr

Abdelmalek Benzekri
IRIT, UT3
Toulouse, France
abdelmalek.benzekri@irit.fr

***Abstract*—This article presents a novel approach that automates part of the work of network security architects, enabling them to design fine-grained secure network architectures. We have developed a methodology that, starting from high-level security requirements, called intents, and an initial unprotected network architecture, computes the optimal security zones and integrates security functions to protect both inter- and intra-zone communications. We implemented this methodology as a proof-of-concept framework, leveraging the flexibility and expressivity of Answer Set Programming, a form of declarative logic programming.**

***Index Terms*—Network security architecture, Intent-based security, Security automation, Micro-segmentation, Zero-trust architecture, Answer set programming.**

I. Introduction

Modern security strategies, such as defense-in-depth or zero-trust architecture, commonly recommend network zoning, also known as network segmentation. It enhances asset control and minimizes the impact of a cybersecurity breach. For instance, the last version of NIST SP800-160 [1] requires the definition of *enclaves, segments, micro-segments, or other restricted types of resource sets based on criticality and trustworthiness*. Segmentation is one of the six domains of the Cisco Secure Architecture for Everyone framework [2] and, in general, zero-trust architecture aims to achieve dynamic micro-segmentation [3].

As modern networks evolve increasingly in complexity, designing network security architectures isf becoming a significant challenge. According to Cisco [4], the estimated number of connected devices in 2023 was 29.3 billion, with network traffic growing exponentially yearly. This rapid evolution makes it extremely difficult for human security architects to manually oversee the entire landscape, especially when securing critical systems. Protecting such networks requires precision and comprehensive visibility into all network components, yet manual approaches fall short of addressing these demands. While the literature includes several works on network security zoning, [5], [6], [7], [8], these primarily offer general guidelines that often require manual adaptation by security architects.

This work was partially supported by ICO, Institut Cybersécurité Occitanie, funded by Région Occitanie, France, and by the European research projects H2020 LeADS (GA 956562), Horizon Europe DUCA (GA 101086308), ARN TrustInClouds, and CNRS IRN EU-CHECK.

To address these challenges, we propose an approach using intent-based modeling. Intents are high-level security requirements from the human operator's perspective, such as "guarantee a high level of confidentiality between entity x and entity y", which are then automatically translated into specific security functions integrated into the network. While we adhere to the original definition of intent as a goal, existing intent-based languages [9], [10] work at lower levels and fail to capture the abstraction required for effective network security management. These languages are imperative and used to describe low-level technological constraints (e.g., they are used to explicitly force some security protocol or function). Therefore, we designed our intent-based meta-model to bridge this gap.

Translating goal-oriented intents into actionable security implementations is far from trivial since it requires intelligence and expertise. Our approach provides an automated security engineering framework capable of suggesting optimal network zoning and security functions. We implemented our methodology in Answer Set Programming (ASP) [11], a form of logic programming, to ensure that the generated solutions meet all specified intents and are also explainable[1]. Our proof-of-concept tool leverages Python[2], clingo[3] as the ASP grounder and solver, offers an interactive Dash[4] front-end, and supports models written in cuddly Document Language (KDL)[5]. Most of the rules (described in Section V) were written in ASP, while a minority in Python for convenience. We based the writing of our logic rules on best practices for critical systems recommended by experienced security architects [12] and consulted with them to obtain feedback and validate our results.

The contributions of our article are:

1) an approach for describing the security requirements of a network as goal-oriented intents;
2) a methodology for automatically computing secure network architectures able to suggest security zones and place security functions;
3) a tool based on Answer Set Programming (ASP) that implements the whole methodology.

[1]Several open-source tools exist to generate an explanation for a solution to an ASP problem. For instance, `viasp` (https://github.com/potassco/viasp).
[2]https://www.python.org/
[3]https://potassco.org/
[4]https://dash.plotly.com/
[5]https://kdl.dev/

The remainder of the article is structured as follows. We introduce a motivation example in Section II. Section III gives an overview of our approach. In Section IV, we describe the mathematical formalism of our meta-model, while Section V details our methodology. Section VI discusses the related works, and finally, we conclude with some future research directions in Section VII.

II. Motivating example

To clarify some key concepts, we will introduce a realistic, simple, yet not trivial example that will be used throughout this paper. Fig. 1 contains a graphical representation of our example's landscape. We define as *landscape* the network (or networks) we want to configure and, potentially, their neighboring networks, such as the Internet.

In this scenario, a company has two separate branches (A and B) in two distinct geographical locations. Every branch has its network, and they communicate via the Internet. These two networks have no security (yet), and our goal is to redesign them both, injecting a proper level of security.

The two networks are almost identical, and both contain:

- some clients used by the local employees;
- a web server containing the company website;
- an app server used by the local employees to perform some accounting operations;
- a DataBase (DB) server leveraged by the app server to store the accounting data;
- a Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) servers.

For performance reasons, the branch A network also has a load balancer that dispatches the external visitors' web traffic between the web servers located in the two branches. In addition, the two DB servers communicate constantly to synchronize their data. The landscape alone does not contain enough information to generate a secured version of itself since it is only a network topological description. In our approach, the intents provide this additional security-related semantics. An *intent* is a high-level descriptive policy declaring the security requirements over a connection. Our approach is flexible enough to accommodate a variety of security requirements, but in our example, we are interested only in three:

- *authorization*, which specifies that two end-points can (bidirectionally) exchange data;
- *confidentiality*, which restricts the access to the exchanged data to only the two communication end-points;
- *integrity*, which ensures that the exchanged data is not altered during its transmission.

In our scenario, we can split all the intents into three groups.

Website-related intents: These intents authorize the employees and the visitors to access the website (reachable via the load balancer and the two web servers). They require low confidentiality and integrity since the website is public and does not contain critical data.

Infrastructure-related intents: These intents enable access to the DHCP and DNS servers for all the nodes in their respective networks. Access to these network functions is essential to the correct functioning of the network, so we want a higher level of integrity to avoid data corruption or manipulation. On the other hand, they only require low confidentiality since the exchanged data is not sensitive.

Accounting-related intents: These intents relate to the company's core accounting functions. They allow access to the app server by the employees and the DB servers but also authorize the synchronization operations between these two DB servers. These intents require maximum confidentiality and integrity since the transmitted information is critical to the company.

Knowing all this information, albeit written more formally, our approach can automatically reconfigure the two branch networks by reorganizing the topology and strategically deploying the most appropriate security functions in the right place while respecting the security requirements expressed by the intents. The following sections will detail how our approach can automatically produce a new secured network architecture.

III. Overview of our approach

The core concept we leverage in securing a network is the notion of *zone*, which is a group of (security) functions with equivalent security requirements (expressed by the intents).

Our approach exploits the zone paradigm, and Fig. 2 shows its high-level workflow. The input of our logic-based engine is a structure we call *realm*. An input *realm* combines an initial *unprotected* landscape, describing a topology where we want to inject security, and an intent Knowledge Base (KB), containing the intents expressing the security requirements. Our *engine* analyzes the unprotected landscape and the intent KB, and it generates a new realm containing a *protected* landscape satisfying all the security constraints expressed by the intents in a (potentially new) intent KB. Note that the output intent KB might differ from the input one; we will explain the reasons in Section V. This process runs in multiple consecutive stages:

I) *micro-segmentation*: the network functions are grouped into security zones;
II) *intra-zone design*: the internal architecture of each zone is computed;
III) *inter-zone design*: the zones are topologically sorted from the highest to the lowest critical and connected.

In Section IV, we will describe the meta-model containing all the information the system needs to operate, while in Section V, we will discuss in detail how the logic engine works and its three stages.

IV. Meta-model

This section describes the mathematical formalism of our meta-model, which can express realms, landscapes, intents, and all their sub-structures.

A *realm* R represents both the input and output of our system (see Fig. 2). A realm is a tuple $R = (L, K)$ containing two elements: a landscape L, a model of the network (or

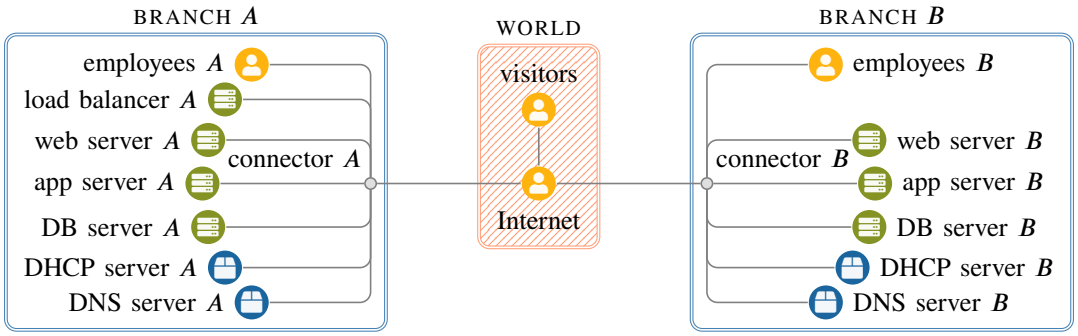

Fig. 1. A sample of initial unprotected landscape.

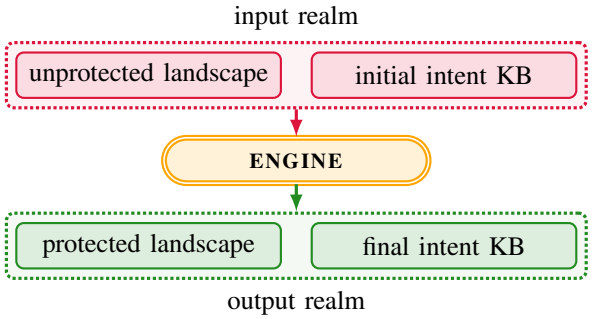

Fig. 2. The workflow of our approach.

networks) to analyze and protect, and an intent knowledge base K, containing information about the intents.

A *landscape* $L = \left(\{ D_i \}_i, \{ C_j \}_j \right)$ represents a generic system of interconnected networks, such as graphically shown in Fig. 1. A landscape not only contains the networks to be secured but can also contain other networks, such as the Internet or some subnet outside our control, such as some external website visitors. It is a tuple containing a set of domains $\{ D_i \}_i$ and channels $\{ C_j \}_j$.

A *domain* $D = \left(d, \{ Z_i \}_i, \{ F_j \}_j \right)$ represents a network under the control of a single administrative entity (e.g., a company, an institution, or a private individual). For instance, in Fig. 1, there are three domains: branch A, branch B, and world. Each domain is considered an independent entity from the others. All domains have a type d that can be *internal* d^{int}, indicating that we have control over that network (e.g., the branch A and B domains), or *external* d^{ext}, indicating a domain outside our jurisdiction (e.g., the world domain). In addition, it contains a set of zones $\{ Z_i \}_i$ and network functions $\{ F_j \}_j$.

A *security zone* $Z = \{ F_i \}_i$ represents a subset of a domain and is just a collection of network functions with equivalent security requirements. In an unprotected landscape, there are no security zones; they are added later by our engine.

A *network function* $F = (f, \lambda)$ represents a network node, physical or virtual. A function is a pair of a type f and a unique identifier λ. We have six types of functions:

- *business* functions f^{bus}, which represent a node with predictable and well-defined behaviors, such as most of the servers — they are the green circles in Fig. 1 (i.e., the load balancer, web, app, and DB servers);
- *environment* functions f^{env} model an entity that is either uncontrollable or with undefined behaviors, such as clients or the Internet itself — Fig. 1 depicts them as yellow circles (i.e., the Internet, the visitors, and employees);
- *infrastructure* functions f^{inf} are similar to business functions; that is, they too have well-defined behavior, but in addition, these functions can be replicated and split into smaller copies, such as a DNS server and other common networking facilities — they are blue circles in Fig. 1 (i.e., the DHCP and DNS servers);
- *connector* functions f^{con} represent a network node that can redirect the traffic to multiple destinations such as routers, switches, and hubs — they are the gray circles in Fig. 1 (i.e., the connectors A and B);
- *split* functions f^{spl} are automatically generated by our system, and they do not appear in the input landscape and represent the split version of an f^{inf} function (see Section V);
- *security* functions f^{sec}, finally, are the security functions themselves, and our logic-based approach deploys them automatically in the right places. By writing the appropriate logic rules, our approach can support any security function; in this paper, we are only interested in:
  - Packet Filter (PF) functions f^{pf} such as iptables;
  - Application Layer Filter (ALF) functions f^{alf} such as ModSecurity for web servers;
  - Virtual Private Network (VPN) terminator functions f^{vpn} such as strongSwan.

Besides domains, zones, and functions, a landscape also contains a set of channels. A *channel* $C = \{ F_1, F_2 \}$ represents a bidirectional connection between two functions F_1 and F_2, such as a cable. In Fig. 1, they are shown as solid gray lines.

An *intent KB* $K = \left(\{ P_i \}_i, \{ I_j \}_j \right)$ contains all the information needed to inject the desired security into a landscape. It is a tuple of profiles $\{ P_i \}_i$ and intents $\{ I_j \}_j$.

An *intent profile* $P = \{ r_i \}_i$ is a tuple of security requirements $(r_i)_i$ and represents a way of organizing and giving a more meaningful structure to the intents themselves.

For instance, in Section II, we grouped the intents into three categories (website, infrastructure, and accounting-related intents). Each category is a profile. A security requirement $r \in \mathbb{N}_{\geq 0}$ is simply an integer value where 0 means that we do not care about that requirement, and the higher the value, the more critical the requirement. Our approach can accommodate any security requirement; however, we are only interested in *confidentiality* r^{con} and *integrity* r^{int} in this context, so that $P = \left(r^{\text{con}}, r^{\text{int}} \right)$. Table I shows the profiles in our example.

NAME	r^{con}	r^{int}
website access	1	1
infrastructure access	1	2
accounting access	3	3

TABLE I
PROFILES OF OUR EXAMPLE.

Finally, an *intent* $I = \left(P, \left\{ F_1, F_2 \right\} \right)$ associates a profile P to a pair of functions F_1 and F_2, acting as the endpoint of the communication. The two network functions effectively specify the *authorization* requirement of the communication while the associated profile expresses all the other security requirements. In our meta-model, we assume that all the intents specify a bidirectional connection. The difference between a channel and an intent is that channels are low-level transmission media without requirements. On the other hand, intents have security constraints and are about reachability so that an intent can use multiple channels at a lower level. Table III in Appendix A contains all the intents we defined in our example.

We will introduce the notation of *recursive membership* $\overset{\star}{\in}$ to simplify some formulas. The notation $x \overset{\star}{\in} X$ indicates that x is either a direct member of X or a member of its recursive elements if X is a set or tuple. Formally:

$$x \overset{\star}{\in} X \implies \begin{cases} x \in X & \text{if } x \text{ is a direct member of } X \\ \exists X_i : x \overset{\star}{\in} X_i & \text{if } X = \left\{ X_i \right\}_i \text{ or } X = \left(X_i \right)_i \end{cases}.$$

For instance, $F_1 \overset{\star}{\in} K$ is true when F_1 is an endpoint of an intent $I = \left(P, \left\{ F_1, F_2 \right\} \right)$ and that, in turn, I is contained in an intent KB $K = \left(\left\{ P_i \right\}_i, \left\{ I_1, \dots \right\} \right)$.

V. OUR APPROACH

Our logic-based engine has three consecutive stages, which we will explain in detail in the following paragraphs. For brevity's sake, we will omit some formula definitions when they are trivial or well-known in the scientific literature. Fig. 3 shows the protected sample landscape generated by our approach. We will indicate with an empty circle accent an object related to the input unprotected realm, such as $\mathring{R}$, $\mathring{L}$, and $\mathring{I}$. On the other hand, we will denote with a filled circle accent a newly created structure related to the output protected realm that each stage will incrementally build, such as $\bullet{R}$, $\bullet{L}$, and $\bullet{I}$.

Stage I: micro-segmentation

This stage computes the security zones (see Section IV). Its output will be a realm $\bullet{R}$ consisting of a landscape $\bullet{L}$ with the functions grouped into zones. However, $\bullet{L}$ only contains the channels inside the external domains; the subsequent stages will deploy the channel involving the internal domains.

Preparation

First, the system computes the function (graph) reachability on $\mathring{L}$. We denote $\mathring{F}_1 \leftrightarrow \mathring{F}_2$ to indicate that the functions $\mathring{F}_1$ and $\mathring{F}_2$ are mutually *reachable*, indicating that a sequence of channels connecting them exists. Our approach computes these relationships to maintain the function reachability in the protected landscape $\bullet{L}$ to avoid breaking the unprotected landscape functionalities.

In addition, this stage also computes the security requirements for every function $\mathring{F}$. We will denote this with the symbol $\text{req}\left(\mathring{F}\right) = \left(\left\{ \mathring{r}_{i,j} \right\}_j \right)_i$, that is the result of this function is a tuple of integer sets. In our scenario, we are only interested in confidentiality and integrity, so $\text{req}\left(\mathring{F}\right) = \left(\left\{ \mathring{r}_i^{\text{con}} \right\}_i, \left\{ \mathring{r}_j^{\text{int}} \right\}_j \right)$. The key idea for computing the function requirements is propagating the profile requirements to the intent endpoints. For instance, five intents involve the function web server A (see Table III), and these intents use all three profiles (see Table I). By collecting all the profile requirements, we can deduce that the web server A requirements are $(\{1,3\}, \{1,2,3\})$. We can then perform a reduction operation on each integer set to simplify them; this choice will impact the zone generation. We defined three (user-selectable) approaches:

- the *strict* approach uses the identity function (i.e., is no reduction at all): this method tends to generate smaller security zones, favoring high security and penalizing the network throughput;
- the *range* approach computes the minimum and maximum value for each integer set: this strategy creates bigger zones than the strict approach — for instance, in the case of web server A, it will return $(\{1,3\}, \{1,3\})$;
- the *max* approach calculates the maximum value for each integer set: it generates even larger zones, decreasing the security in favor of network speed — for the web server A, it will produce $(3,3)$.

Our approach can now create the security zones; the function types dictate the method.

Connectors

Connectors will not become part of any zone since we completely rewire the protected landscape's topology.

Business and environment functions

Business and environment functions are grouped into newly created zones using the following logic rules. Since these functions are 'moved' into a zone, $\bullet{F} = \mathring{F}$ holds. This stage computes a series of $\text{inZone}(\mathring{F}_1, \mathring{F}_2)$ atoms, indicating that $\mathring{F}_1$ and $\mathring{F}_2$ belong to the same zone. We want to generate the most extensive zones possible since having two zones with the same requirements in the same domain does not increase the security but only decreases the network throughput. We

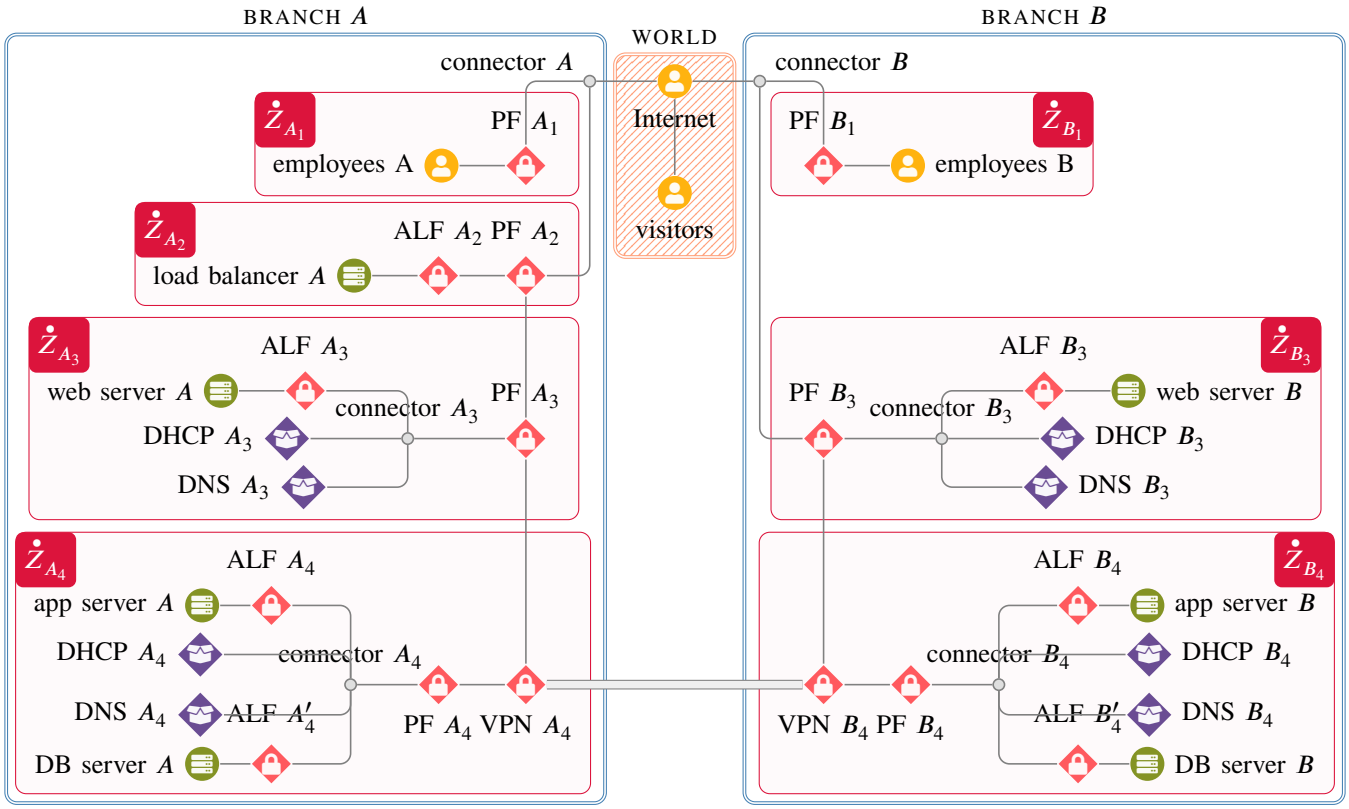

Fig. 3. The sample protected landscape.

can model this as an optimization problem with the following objective function:

$$\max \left| \left\{ \text{inZone}\left(\mathring{F}_1, \mathring{F}_2\right) : \mathring{F}_1, \mathring{F}_2 \mathring{\in}^{\star} \mathring{R} \right\} \right|.$$

The following paragraphs introduce the constraints.

Internal domain constraint: All the functions in a zone must belong to the same internal domain.

$$\text{inZone}\left(\mathring{F}_1, \mathring{F}_2\right) \wedge \mathring{F}_1, \mathring{F}_2 \mathring{\in} \mathring{D} \wedge d^{\text{int}} \in \mathring{D}.$$

Reachability constraint: All the functions in a zone must be reachable in the original unprotected landscape.

$$\text{inZone}\left(\mathring{F}_1, \mathring{F}_2\right) \wedge \mathring{F}_1 \leftrightarrow \mathring{F}_2.$$

Type homogeneity constraint: All the functions in the same zone must have the same type, which can be f^{bus} or f^{env}. This strategy ensures that business and environment functions are not mixed. We consider environment functions less trustworthy since they are not predictable. In other words, this ensures that clients and servers live in separate zones.

$$\text{inZone}\left(\mathring{F}_1, \mathring{F}_2\right) \wedge \left(\left(f^{\text{bus}} \in F_1 \wedge f^{\text{bus}} \in F_2\right) \vee \left(f^{\text{env}} \in F_1 \wedge f^{\text{env}} \in F_2\right)\right).$$

Security homogeneity constraint

All the functions in a zone must have the same security requirements.

$$\text{inZone}\left(\mathring{F}_1, \mathring{F}_2\right) \wedge \text{req}\left(\mathring{F}_1\right) = \text{req}\left(\mathring{F}_2\right).$$

We can find the zones by gathering all the inZone atoms. For instance, on our sample network with the strict approach, the system groups the business and environment functions into the zones listed in Table II.

ID	FUNCTIONS	r^{con}	r^{int}
$\dot{Z}_{A_1}$	employees A	1, 3	1, 3
$\dot{Z}_{A_2}$	load balancer A	1	1
$\dot{Z}_{A_3}$	web server A	1, 3	1, 2, 3
$\dot{Z}_{A_4}$	app server A and DB server A	1, 3	2, 3
$\dot{Z}_{B_1}$	employees B	1, 3	1, 3
$\dot{Z}_{B_2}$	web server B	1, 3	1, 2, 3
$\dot{Z}_{B_3}$	app server B and DB server B	1, 3	2, 3

TABLE II
ZONES IN OUR EXAMPLE.

Infrastructure functions

Treating infrastructure functions like business and environment functions and moving them into a zone is risky. If they get compromised, an attacker can indirectly taint all their connected functions, potentially affecting the entire domain. To enhance security, we clone and split these functions for each zone so that if a split infrastructure function is compromised, only its containing zone will be compromised.

Split infrastructure function generation rule: We will create a new function $\dot{F}$ of type f^{spl} every time the following condition applies:

$$f^{\text{inf}} \in \mathring{F}_1 \wedge \left(f^{\text{bus}} \in \mathring{F}_2 \vee f^{\text{env}} \in \mathring{F}_2\right) \wedge \mathring{F}_2 \in \dot{Z}_2 \wedge$$
$$\mathring{F}_1, \mathring{F}_2 \overset{\star}{\in} \mathring{I} \implies f^{\text{spl}} \in \dot{F}^{\text{spl}}_{\left(\mathring{F}_1, \mathring{F}_2\right)} \wedge \dot{F}^{\text{spl}}_{\left(\mathring{F}_1, \mathring{F}_2\right)} \in \dot{Z}_2.$$

A split infrastructure function links an infrastructure function (hence denoted as the *source* function) to a non-infrastructure function (the *target* function) via an intent. Note that a split infrastructure function is located in the zone of its target function. For instance, in our example, $\dot{Z}_{A_1}$ will contain three DHCP servers: one for the web server, and the other two for the app and DB servers. Once we find all the split infrastructure functions, we can do a quick optimization pass and merge all functions in the same zone and with the same source. For instance, in $\dot{Z}_{A_4}$ we will have two clones of DHCP server A: one for managing the app server A and another one for managing the DB server A. We can merge these two functions since it will not impact the system's security and simplify its deployment. We can then delete the original infrastructure version of an infrastructure node since they are no longer needed. Since the new landscape $\dot{L}$ will not contain any infrastructure function (only split ones), we need to update all the original intents by replacing the deleted endpoints with their new split infrastructure counterparts (we omit the formulas for brevity), thus producing a new intent KB $\dot{K}$.

Stage II: intra-zone design

In the second stage, our approach restructures each zone independently by adding the security functions and the internal channels.

Preparation

As a preparatory step, we compute $\text{req}\left(\dot{Z}\right)$, denoting the security requirements of a zone $\dot{Z}$. It can be formally defined as: $\text{req}\left(\dot{Z}\right) = \text{req}\left(\mathring{F}\right)$ where $\mathring{F} \in \dot{Z}$.

Security functions

The following paragraphs show the rules for placing the security functions.

PF placement rule: PF functions are helpful and cheap, so we put one in each zone with at least one non-null requirement.

$$\min\left(\text{req}\left(\dot{Z}\right)\right) > 0 \implies f^{\text{pf}} \in \dot{F}^{\text{pf}}_{Z} \wedge \dot{F}^{\text{pf}}_{Z} \in \dot{Z}.$$

ALF placement rule: We create an ALF function for each business function to protect them against data exfiltration towards other zones (i.e., when the confidentiality requirement is positive), but also for corruption when the data arrives from other zones (i.e., when the integrity requirement is positive).

$$f^{\text{bus}} \in \mathring{F} \wedge \mathring{F} \in \dot{Z} \wedge \exists \mathring{r}^{\text{con}} > 0 : \mathring{r}^{\text{con}} \overset{\star}{\in} \text{req}\left(\dot{Z}\right) \wedge$$
$$\exists \mathring{r}^{\text{int}} > 0 : \mathring{r}^{\text{int}} \overset{\star}{\in} \text{req}\left(\dot{Z}\right) \implies f^{\text{alf}} \in \dot{F}^{\text{alf}}_{\mathring{F}} \wedge \dot{F}^{\text{alf}}_{\mathring{F}} \in \dot{Z}.$$

VPN terminator placement rule: VPNs terminators create a sort of distributed zone spanning multiple domains. We create two VPN terminators between two zones if they have the same security requirements and at least one intent connects them with some positive integrity requirement.

$$\mathring{F}_1 \overset{\star}{\in} \dot{Z}_1 \wedge \mathring{F}_2 \overset{\star}{\in} \dot{Z}_2 \wedge Z_1 \neq Z_2 \wedge$$
$$\mathring{F}_1, \mathring{F}_2 \overset{\star}{\in} \mathring{I} \wedge \mathring{r}^{\text{int}} > 0 \wedge \mathring{r}^{\text{int}} \overset{\star}{\in} \mathring{I} \wedge \text{req}\left(\dot{Z}_1\right) = \text{req}\left(\dot{Z}_2\right) \implies$$
$$f^{\text{vpn}} \in \dot{F}^{\text{vpn}}_{\dot{Z}_1, \dot{Z}_2} \wedge \dot{F}^{\text{vpn}}_{\dot{Z}_1, \dot{Z}_2} \in \dot{Z}_1 \wedge \dot{F}^{\text{vpn}}_{\dot{Z}_2, \dot{Z}_1} \wedge \dot{F}^{\text{vpn}}_{\dot{Z}_2, \dot{Z}_1} \in \dot{Z}_2$$

For instance, in our example, we can create two VPN terminators to connect the zones $\dot{Z}_{A_4}$ and $\dot{Z}_{B_4}$. These zones have the same security requirements, and the two DB servers will use this tunnel to communicate.

Connectivity

After the system places the security functions, the next task is to insert some channels to allow communication. The channel placement inside a zone is relatively unimportant from a security point of view since all the functions inside a zone trust each other. The following paragraphs describe how we create the channels:

1) we create a channel between each ALF function and the function they protect;
2) if there is more than one business or environment function, we create a connector and link it with all the ALF functions and all the non-ALF-protected functions;
3) if there is a PF function, we create a channel between it and: a) the connector, if it exists; b) otherwise the only ALF function, if it exists; c) otherwise the only business or environment function;
4) if there is a VPN terminator function but no PF, we follow the same strategy described in Item 3;
5) if there is both a PF function and a VPN terminator, create a channel between them.

Fig. 3 depicts the final design of the zones.

Stage III: inter-zone design

The final stage consists of topologically sorting the zones in increasing order of criticality and connecting them. More formally, the objective of this stage is to construct a *zone forest* (a set of *zone trees*) where the vertexes are the zones and the edges are the channels connecting the zones. In this context, we define a *DeMilitarized Zone (DMZ)* as a zone with a channel spanning multiple domains. In a zone tree, the root is always a DMZ. A domain might contain multiple trees since our approach can suggest using multiple DMZs to increase its security. Since we are dealing with graphs, we will use the notations $\text{depth}(Z)$ and $\text{parent}(Z)$ to indicate respectively the depth of a zone Z and its parent. As a rule of thumb, each domain does not trust the others, so the closer a zone is to another domain, the less protected it is. That means that DMZs are the unsafest zones, while the tree leaves are the most protected ones.

Preparation

Before creating the tree forests, the system computes the domain adjacency of $\mathring{L}$. We indicate that $\mathring{D}_1$ and $\mathring{D}_2$ are adjacent with the notation $\mathring{D}_1 \sim \mathring{D}_2$, that is, at least one channel connects the two domains. The protected landscape must respect the unprotected domain adjacencies. This behavior is essential to avoid situations such as placing a channel directly connecting the two branches and bypassing the Internet.

Zone precedences

Before effectively establishing an order of the zones, we compute the allowed precedences. We will indicate with $\dot{Z}_1 \succ \dot{Z}_2$ the fact that $\dot{Z}_1$ can be *preceded* by $\dot{Z}_2$, that is $\dot{Z}_1$ has higher security requirements than $\dot{Z}_2$. That also means that $\dot{Z}_1$ can have $\dot{Z}_2$ as a parent. We can use the CANPRECEDE function (see Algorithm 1) to check the precedences between two zones.

Input: two zones $\dot{Z}_1$ and $\dot{Z}_2$
Output: a boolean value stating if $\dot{Z}_1 \succ \dot{Z}_2$

$\left(\bar{r}_{1,1}, \dots, \bar{r}_{1,n}\right) \leftarrow \text{req}(\dot{Z}_1)$
$\left(\bar{r}_{2,1}, \dots, \bar{r}_{2,n}\right) \leftarrow \text{req}(\dot{Z}_2)$
for $i \leftarrow 1$ **to** n **do**
  $in \leftarrow \bar{r}_{1,i} \cap \bar{r}_{2,i}$
  $out \leftarrow \bar{r}_{2,i} \setminus \bar{r}_{1,i}$
  if $|in| > 0$ **then** $l_{in} \leftarrow \min(in) = \min(\bar{r}_{1,i})$
  else $l_{in} \leftarrow \top$
  if $|out| > 0$ **then** $l_{out} \leftarrow \max(out) < \min(\bar{r}_{1,i})$
  else $l_{out} \leftarrow \top$
  if $\bar{r}_{1,i} \neq \bar{r}_{2,i} \wedge (l_{in} \vee l_{out})$ **then** **return** $\bot$
end
return $\top$

Algorithm 1: The CANPRECEDE function.

In Algorithm 1, we denote the false and true atoms with $\bot$ and $\top$. Intuitively, the CANPRECEDE function iterates over all the requirement sets for the two zones and performs some parallel comparisons (e.g., it compares the confidentiality and integrity pairs of the two zones). For each integer set pair, it checks if we are in one of the following cases:

- the two sets are identical;
- the two sets are disjoint, and one set is strictly lower than the other one (e.g., $\{1,2\}$ and $\{3,4,5\}$);
- the sets are not disjoint, and they overlap without gaps if we treat them as sorted tuples — for instance, $\{1,2,3\}$ and $\{2,3,4\}$ overlap without gaps, but $\{1,3\}$ and $\{2,3,4\}$ do not perfect overlap (there is a spurious 2).

For instance, $\dot{Z}_{A_4} \succ \dot{Z}_{A_3}$. The confidentiality requirements of the zones are both $\{1,3\}$, while the integrity requirements are $\{2,3\}$ and $\{1,2,3\}$, which overlap without gaps. Note also that the zones with the lowest number of precedences are *candidate DMZs* since the lower the number of preceding zones, the less stringent the security. We will indicate that a zone $\dot{Z}$ is a candidate DMZ with $\text{candidateDMZ}(\dot{Z})$.

Zone distances

Given two zones $\dot{Z}_1$ and $\dot{Z}_2$ and their requirements $\text{req}\left(\dot{Z}_1\right) = \left(\bar{r}_{1,i}\right)_i$ and $\text{req}\left(\dot{Z}_2\right) = \left(\bar{r}_{2,i}\right)_i$, we can compute the zone *distance* as the value:

$$\text{distance}\left(\dot{Z}_1, \dot{Z}_2\right) = \sum_i \left(\omega_i \left| \sum \bar{r}_{1,i} - \sum \bar{r}_{2,i} \right|\right).$$

Our formula is a Manhattan-weighted distance where the ω_i coefficients are the user-defined weights for the requirements. The bigger the distance between two zones, the more different their security requirements. In our scenario, for example, if we assume to have a confidentiality weight of 1 and an integrity weight of 2, we can compute the distance between $\dot{Z}_{A_4}$ and $\dot{Z}_{A_3}$ as $1 \cdot |(1+3) - (1+3)| + 2 \cdot |(2+3) - (1+2+3)| = 2$.

Forest construction

We model the problem of topologically sorting the zones and building the forests as a lexicographic multi-objective optimization problem. In our approach, we aim to minimize three functions. Their order can be customized, but we present them here in the default order:

$$\text{lexmin} -f_{\text{bus}}\left(\dot{L}\right), f_{\text{dis}}\left(\dot{L}\right), f_{\text{env}}\left(\dot{L}\right).$$

Maximize business zone depth: We want to maximize the tree depth of the zones with some business functions. This behavior ensures that the business functions are well protected since the deeper a zone in a tree, the safer it is.

$$f_{\text{bus}}\left(\dot{L}\right) = \max\left\{\text{depth}\left(\dot{Z}\right) : \exists \mathring{F} \in \dot{Z} \wedge f^{\text{bus}} \in \mathring{F} \wedge \dot{Z} \dot{\in}^{\star} \dot{L}\right\}.$$

Minimize adjacent zone distances: We want adjacent zones to have similar requirements to avoid too many security jumps.

$$f_{\text{dis}}\left(\dot{L}\right) = \min\left\{\text{distance}\left(\dot{Z}_1, \dot{Z}_2\right) : \exists \dot{F}_1 \in \dot{Z}_1 \wedge \exists \dot{F}_2 \in \dot{Z}_2 \wedge \dot{F}_1, \dot{F}_2 \in \dot{C} \wedge \dot{C} \dot{\in}^{\star} \dot{L}\right\}.$$

Minimize environment zones depth: We place the zones with some environment functions closer to the tree roots. Environment functions are nonpredictable, hence untrustworthy, so we want them closer to the DMZs.

$$f_{\text{env}}\left(\dot{L}\right) = \min\left\{\text{depth}\left(\dot{Z}\right) : \exists \mathring{F} \in \dot{Z} \wedge f^{\text{env}} \in \mathring{F} \wedge \dot{Z} \dot{\in}^{\star} \dot{L}\right\}.$$

To simplify some formulas, we will use the function $\text{endpoint}(Z)$ or $\text{endpoint}(D)$ to indicate a suitable function as the endpoint of a new channel. For external domains, the suitable endpoints are the endpoints used by the inter-domain channels. In the case of zones, this function will return in order the VPN terminator, PF function, connector, ALF function, or the sole function, depending if one of these functions exists.

The following paragraphs describe the constraints.

DMZs selection constraint: This constraint ensures that DMZs are selected from the candidate DMZs.

$$\forall\, \text{DMZ}\left(\dot{Z}\right) : \text{candidateDMZ}\left(\dot{Z}\right).$$

DMZs separation constraint: DMZs in the same domain must not connected by a channel since they are the roots of separate trees.

$$\nexists \dot{C} : \dot{F}_1, \dot{F}_2 \in \dot{C} \wedge \dot{F}_1 \in \dot{Z}_1 \wedge \dot{F}_2 \in \dot{Z}_2 \wedge \dot{Z}_1 \neq \dot{Z}_2 \wedge \dot{Z}_1, \dot{Z}_2 \overset{\star}{\in} \dot{D} \wedge \mathrm{DMZ}\left(\dot{Z}_1\right) \wedge \mathrm{DMZ}\left(\dot{Z}_2\right).$$

DMZ existence constraint: If there is a channel between two domains and at least one is internal, this channel must have an endpoint in a DMZs.

$$\dot{F}_1, \dot{F}_2 \in \dot{C} \wedge \dot{F}_1 \in \dot{Z}_1 \wedge \dot{Z}_1 \overset{\star}{\in} \dot{D}_1 \wedge \dot{F}_2 \overset{\star}{\in} \dot{D}_2 \wedge \dot{D}_1 \neq \dot{D}_2 \wedge \mathrm{DMZ}\left(\dot{Z}_1\right).$$

Intra-domain channel constraint: A non-DMZ can only have channels inside its domain.

$$\dot{F}_1, \dot{F}_2 \in \dot{C} \wedge \dot{F}_1 \in \dot{Z}_1 \wedge \dot{F}_2 \in \dot{Z}_2 \wedge \dot{Z}_1 \neq \dot{Z}_2 \wedge \dot{Z}_1, \dot{Z}_2 \overset{\star}{\in} \dot{D} \wedge \neg\,\mathrm{DMZ}\left(\dot{Z}_1\right) \wedge \neg\,\mathrm{DMZ}\left(\dot{Z}_2\right).$$

Authorization requirement constraint: The endpoints of an intent must be reachable.

$$\dot{F}_1, \dot{F}_2 \overset{\star}{\in} \dot{I} \wedge \dot{F}_1 \nleftrightarrow \dot{F}_2.$$

Endpoint constraint: New channels are created only between unauthorized endpoints.

$$\dot{F}_1, \dot{F}_2 \in \dot{C} \wedge \dot{F}_1 \in \dot{Z}_1 \wedge \dot{F}_1 = \mathrm{endpoint}\left(\dot{Z}_1\right) \wedge \dot{F}_2 \in \dot{Z}_2 \wedge \dot{F}_2 = \mathrm{endpoint}\left(\dot{Z}_2\right).$$

Domain adjacency constraint: The unprotected landscape's domain adjacency must be preserved.

$$\mathring{\dot{D}}_1 \sim \mathring{\dot{D}}_2 \wedge \dot{D}_1 \sim \dot{D}_2$$

Zone precedence constraint: A security zone can only be preceded by a lower security zone in its tree.

$$\dot{F}_1, \dot{F}_2 \in \dot{C} \wedge \dot{F}_1 \in \dot{Z}_1 \wedge \dot{F}_2 \in \dot{Z}_2 \wedge \dot{Z}_1 \neq \dot{Z}_2 \wedge \dot{Z}_1 \succ \dot{Z}_2.$$

Forest constraint: The zone graph must be a forest; all the zones must have at most one parent.

$$\nexists\, \mathrm{parent}(\dot{Z}) \vee \exists!\, \mathrm{parent}(\dot{Z}).$$

VI. Related works

Several researchers [13], [14] emphasized the potential of intent in driving security automation. Nevertheless, a recent survey [15] concluded that "most of the automated methodologies for configuring network security services focus on a single function type, with packet filtering firewall being the most dominant one".

Governmental agencies and industrial forums have published guidelines for building secure networks. The Canadian Centre for Cyber Security [5] has developed a network security zone reference model, which proposes seven generic zone categories (public zone, public access zone, operations zone, restricted zone, highly restricted zone, restricted extranet zone and management zone) which are interconnected through dedicated interface points. The guidelines include a generic model and specify the requirements for zones. The British Columbia model [6] proposes also 7 zones and allows communication inside the zones and only between adjacent zones. Secure Arc [7] defines 8 zones and introduces a cross-zone segmentation concept called "silos". Communication is only allowed between adjacent zones and within the same silo or between adjacent silos within the same zone to restrict inter-zone interaction. Finally, ISA/IEC 62443 [8] specifies requirements for secure industrial automation and control systems. It introduces the concept of zone and conduit, a logical grouping of communication channels between zones sharing common security requirements. These guidelines provide general zoning strategies but require manual customization prone to human error.

The academic community has published few works on network security segmentation. Gontarczyk et al. [16] proposed a standard blue-print that includes three classes of security zone (no physical measures, limited physical measures, and strong physical measures). It also provides a classifier to guide the deployment of systems/applications. However, these are high-level guidelines that must be manually customized by security architects. Li et al. [17] have presented an algorithm for micro-segmentation in cloud data centres based on VLAN and VxLAN mapping. Kwon et al. [18] focused on inter-zone communication and described a unified security gateway called Zone Translation Point. This research has been extended by Furrer et al. [19] to adapt to industrial networks and comply with ISA/IEC 62443 [8]. Some researchers also proposed a bottom-up approach [20] or risk analysis on existing network security zone models [21], [22]. The closest work to ours is by Laborde et al. [11], where the authors express integrity as integers on nodes to generate high-level network security architectures. Our methodology instead, also considers confidentiality and implements a more comprehensive modeling based on industrial best practices and intents for better flexibility.

VII. Conclusions

We presented a logic-based approach capable of automatically secure a network starting from a set of intents, high-level security requirements without technological constraints. Our approach implements various logic rules based on a set of best practices used by real-world security architects. We implemented a proof-of-concept in Python and clingo to leverage the flexibility and expressiveness of ASP, a declarative logic programming methodology, particularly efficient in solving complex search and combinatorial problems.

Our future work will involve integrating our tool with a Software-Defined Networking (SDN) infrastructure, which will allow us to suggest a secure architecture, deploy it, and dynamically adjust security policies in real-time. We also want to automatically configure the security functions appropriately and, to achieve this, we are exploring the use of the OASIS OpenC2 standard [23].

Acknowledgements

We thank Yves Rütschlé, Product Security Architect at Airbus Protect, for all his valuable feedback and comments.

References

[1] R. Ross, V. Pillitteri, R. Graubart, D. Bodeau, and R. McQuaid, "Developing Cyber-Resilient Systems: A Systems Security Engineering Approach," National Institute of Standards and Technology, Tech. Rep., 2021.

[2] Cisco, "Cisco Secure Architecture for Everyone (SAFE)," Cisco, Tech. Rep., 2023. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/enterprise/design-zone-security/safe-overview-guide.pdf

[3] R. Chandramouli, "Guide to a Secure Enterprise Network Landscape," National Institute of Standards and Technology, Tech. Rep., 2022. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/215/final

[4] Cisco, "Cisco Annual Internet Report (2018–2023)," Cisco, Tech. Rep., 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[5] C. C. for Cyber Security, "ITSP.80.022 Baseline Security Requirements for Network Security Zones," Canadian Centre for Cyber Security, Tech. Rep., 2022. [Online]. Available: https://www.cyber.gc.ca/sites/default/files/cyber/publications/itsp80022-e.pdf

[6] C. Lyons, "Enterprise IT security architecture security zones: Network security zone standards," 2012.

[7] S. Arc, "Secure Arc Reference Architecture," Secure Arc, Tech. Rep., last access 2023. [Online]. Available: https://www.securearc.com/referencearchitecture/#DesignPatterns

[8] I. G. C. Alliance, "Security of Industrial Automation and Control Systems: An Overview of ISA/IEC 62443 Standards," ISA Global Cybersecurity Alliance, Tech. Rep., 2023. [Online]. Available: https://21577316.fs1.hubspotusercontent-na1.net/hubfs/21577316/2023%20ISA%20Website%20Redesigns/ISAGCA/PDFs/ISAGCA%20Quick%20Start%20Guide%20FINAL.pdf

[9] A. S. Jacobs, R. J. Pfitscher, R. H. Ribeiro, R. A. Ferreira, L. Z. Granville, W. Willinger, and S. G. Rao, "Hey, Lumi! Using Natural Language for Intent-Based Network Management," in *Proceedings of USENIX ATC 2021: USENIX Annual Technical Conference*, 2021.

[10] S. Hares, "Intent-Based Nemo Overview," Internet Engineering Task Force, Tech. Rep., 2015. [Online]. Available: https://datatracker.ietf.org/doc/draft-hares-ibnemo-overview/01/

[11] R. Laborde, S. T. Bulusu, A. S. Wazan, F. Barrère, and A. Benzekri, "Logic-based methodology to help security architects in eliciting high-level network security requirements," in *Proceedings of SAC 2019: ACM/SIGAPP Symposium on Applied Computing*, 2019.

[12] Y. Rütschlé, "A security model for distributed critical systems," Aribus Protect, Tech. Rep., 2024. [Online]. Available: https://www.protect.airbus.com/insights/white-papers/

[13] I. Ahmad, J. Malinen, F. Christou, P. Porambage, A. Kirstädter, and J. Suomalainen, "Security in Intent-Based Networking: Challenges and Solutions," in *Proceedings of CSCN 2023: IEEE Conference on Standards for Communications and Networking*, 2023.

[14] J. Xu and G. Russello, "Automated Security-focused Network Configuration Management: State of the Art, Challenges, and Future Directions," in *Proceedings of DSA 2022: International Conference on Dependable Systems and Their Applications*, 2022.

[15] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Automation for network security configuration: state of the art and research trends," *ACM Computing Surveys*, 2023.

[16] A. Gontarczyk, P. McMillan, and C. Pavlovski, "Blueprint for Cyber Security Zone Modeling," *Information Technology in Industry*, 2015.

[17] D. Li, Z. Yang, S. Yu, M. Duan, and S. Yang, "A Micro-Segmentation Method Based on VLAN-VxLAN Mapping Technology," *Future Internet*, 2024.

[18] J. Kwon, C. Hähni, P. Bamert, and A. Perrig, "Mondrian: Comprehensive Inter-domain Network Zoning Architecture," in *Proceedings of NDSS 2021: Network and Distributed System Security Symposium*, 2021.

[19] M. S. Furrer, "TABLEAU: Future-Proof Zoning for OT Networks," in *Proceedings of CRITIS 2021: International Conference on Critical Information Infrastructures Security*, 2022.

[20] H. V. Ramasamy, C.-L. Tsao, B. Pfitzmann, N. Joukov, and J. W. Murray, "Towards automated identification of security zone classification in enterprise networks," in *Proceedings of Hoit-ICE 2011: Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, 2011.

[21] H. Holm, K. Shahzad, M. Buschle, and M. Ekstedt, "P2CySeMoL: Predictive, Probabilistic Cyber Security Modeling Language," *IEEE Transactions on Dependable and Secure Computing*, 2014.

[22] R. Mitchell and E. Walkup, "Further refinements to the foundations of cyber zone defense," in *Proceegings of MILCOM 2017: IEEE Military Communications Conference*, 2017.

[23] OASIS, "OpenC2," 2024. [Online]. Available: https://openc2.org/

Appendix A
Intents

Table III lists all the intents used in our example scenario.

ENDPOINT 1	ENDPOINT 2	PROFILE
visitors	load balancer *A*	website access
employees *A*	web server *A*	website access
employees *B*	web server *B*	website access
load balancer *A*	web server *A*	website access
load balancer *A*	web server *B*	website access
web server *A*	DHCP server *A*	infrastructure access
web server *A*	DNS server *A*	infrastructure access
app server *A*	DHCP server *A*	infrastructure access
app server *A*	DNS server *A*	infrastructure access
DB server *A*	DHCP server *A*	infrastructure access
DB server *A*	DNS server *A*	infrastructure access
web server *B*	DHCP server *B*	infrastructure access
web server *B*	DNS server *B*	infrastructure access
app server *B*	DHCP server *B*	infrastructure access
app server *B*	DNS server *B*	infrastructure access
DB server *B*	DHCP server *B*	infrastructure access
DB server *B*	DNS server *B*	infrastructure access
employees *A*	app server *A*	accounting access
employees *B*	app server *B*	accounting access
web server *A*	DB server *A*	accounting access
web server *B*	DB server *B*	accounting access
app server *A*	DB server *A*	accounting access
app server *B*	DB server *B*	accounting access
DB server *A*	DB server *B*	accounting access

TABLE III
INTENTS IN OUR EXAMPLE.