A Framework To Build Secure Microservice Architecture

Wai Yan Elsa Tai Ramirez
University of Texas at El Paso

A FRAMEWORK TO BUILD SECURE MICROSERVICE ARCHITECTURE

WAI YAN ELSA TAI RAMIREZ

Doctoral Program in Computer Science

APPROVED:

Ann Q. Gates, Ph.D., Chair

Salamah I. Salamah, Ph.D., Co-Chair

Jaime Acosta, Ph.D.

Michael Pokojovy, Ph.D.

Stephen L. Crites, Jr., Ph.D.
Dean of the Graduate School

Dedication

To my

Dad, Mom, Sister, and Grand-Parents, and

To the loves of my life:

Jon and Jonathan.

A FRAMEWORK TO BUILD SECURE MICROSERVICE ARCHITECTURE

by

WAI YAN ELSA TAI RAMIREZ, M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

THE UNIVERSITY OF TEXAS AT EL PASO

May 2023

Acknowledgements

First, I want to acknowledge and thank my committee members, Dr. Ann Q. Gates, Dr. Salamah I. Salamah, Dr. Jaime Acosta, and Dr. Michael Pokojovy. Thank you for serving on my committee and for all your help, support, and guidance.

I would like to thank Dr. Gates for being my mentor and for being there for me throughout my entire education. Your feedback and discussions have always been valuable to me, and I appreciate that you would always make time for me no matter how busy your schedule became. I will always hold everything you taught me close in my career going forward.

I would like to thank Dr. Salamah for being my mentor. Your guidance was truly invaluable throughout my work, and I will always appreciate it. Thank you for keeping me focused, encouraging me, and going above and beyond for me. You have been a great mentor and a true friend.

I would like to say a special thank you to my father, Wing Kuen Tai, my mother, Fung Mei Kam, and my sister, Tina Tai. Thank you for always being there for me and supporting me in every journey I have taken. Thank you for all the sacrifices you've made for me so that I could pursue my dreams.

I would like to thank my husband, Jon Ramirez. Thank you for supporting me and for being my pillar throughout this journey. Thank you for always being there for me no matter what happened. You've always encouraged me to believe in myself, fight for what I believe in, and never give up on my dreams. I would not have made it this far if it was not for you.

Abstract

Microservice architecture has become a popular architecture style in recent years. According to a series of surveys conducted by IBM Market Development & Insights in 2021, microservices are heavily used in many industries worldwide. With an increase in the adoption of microservice architecture in the development of applications, such as Netflix, Amazon, Uber, Ebay, Twitter, DoorDash, Capital One, and Monzo, and the increase in security breaches in microservice based systems (e.g., the DoorDash data breaches in 2019 and 2022, Twitter data breach in 2022, and compromises to Netflix's infrastructure), there is a need to examine and understand security issues that exist in microservice architectures.

Security issues within microservice architectures can be summarized with four main points. 1) Security is often considered as an afterthought, rather than during the early development phases. Security considerations are thought of as roadblocks that prevent software from being released on time; 2) There are more vulnerabilities per line of code in applications using microservice architectures compared with equivalent monolithic applications; 3) Microservices present new security challenges that are not present in monolithic applications due to the distributed nature of the architecture; communications between microservices are over the network which means a request may be susceptible to man-in-the-middle attacks; 4) There is a lack of comprehensive knowledge regarding how to build applications using microservice architectures with security in mind.

The goal of the research is two-fold: 1) To study and document security properties that can remediate security issues in microservice architectures; and 2) define an effective approach to assist software architects in formally defining security properties early on in the software development lifecycle.

The research examines microservice security from the perspective of industry and academia. The research questions (RQ) are as follows:

RQ1: What are the security challenges in microservices architecture?

RQ2: What mechanisms are currently used to address the security challenges in microservices architecture?

RQ3: What approach can enhance the security modeling and specification in microservice architectures?

The result of the research is an extensive review of security challenges and practices related to secure microservice architecture that informed the development of a framework that enhances the ability of software architects to formally specify security properties. The resulting framework includes the use of decision trees to guide software architects in determining what specific security properties should be considered, how different security properties are related can be used together, and what additional structural elements (components and connectors) should be considered when adding specific security properties.

The impact of the work is that software vulnerabilities are addressed during early phases of software development (architecture and design) rather than later in the software development lifecycle. This helps to significantly reduce costs associated with software defect mitigation. Studies have shown that the cost ratio in tackling a software defect, including security vulnerabilities, is doubled if defects are discovered during the implementation phase compared to the architecture and design phases. This ratio more than triples if defects are discovered during testing. The work provides comprehensive support in defined security in microservice architectures, especially for software architects who have minimal experience in society.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1. Overview

1.1.1. Motivation

In today's society, software has become an integral part of everyday life and is used in virtually every application domain. In particular, microservice architecture has become a popular architecture style in recent years [9], and according to a series of surveys conducted by IBM Market Development & Insights in 2021 [43], microservices are heavily used in many industries worldwide. It is a popular choice for cloud-based projects due to the scalability in cloud environments and flexibility in software development. Software developers are not limited in the technology used to build each microservice since microservices are integrated via technology-agnostic APIs [23]. Some notable examples of major entities that are using microservice architectures are:

- In 2010, Netflix transitioned from monolithic architecture to microservice architecture. It started using AWS Amazon to host more than 100 grained services [4].
- Monzo, a financial institution in the United Kingdom, announced its microservices deployment. They have more than 1500 services running in its microservices deployment, and they are using network isolation to make their deployment and microservices more secure [26].
- In July 2019, Capital One, one of the leading financial institutions in the United States, announced its microservices deployment. It consists of thousands of microservices on several thousands of containers and thousands of Amazon Elastic Compute Cloud instants [26].

According to the International Data Corporation [34], by 2022, 90% of all apps will feature microservice architectures.

As software becomes more prevalent in day-to-day services and activities, software attacks are increasing in frequency and severity. The following are examples of notable vulnerabilities and attacks that took place between 2016 and 2022:

- In 2017, Equifax [8, 52] had a data breach that affected at least 145.5 million individuals in the U.S. and nearly 1 million people outside the U.S. The breach was caused by a known vulnerability in Apache Struts which allowed unauthorized access to user data maintained by Equifax.
- In May 2019, Doordash [53] disclosed a data breach that exposed 4.9 million users' personal data. The affected personal data included personal information such as names, email addresses, delivery addresses, order history, phone numbers, hashed salted passwords, the last four digits of credit card numbers, the last four digits of bank account numbers, and about 100000 dashers' driver's license numbers. The affected personal data was accessed by an unauthorized third party.
- In August 2022, Doordash [50] announced that one of the third-party vendors that it uses was compromised, and an unauthorized party was able gain access to some of DoorDash's internal tools using the stolen credentials of the third-party vendor's employees. Personal information maintained by DoorDash was affected in this breach. This included name, email address, delivery address, phone numbers, basic order information, and partial payment card information.
- In August 2022, Twitter [21] confirmed that 5.4 million users were affected by a July 2022 data breach. The data breach was caused by a vulnerability in the system which allowed anyone without any authentication to obtain a user's twitter internal identifier by providing a phone number or email address even after the user has disabled this action in the privacy setting [17].
- Netflix [32] experienced a security breach when one of its subdomains was compromised. Adversaries were able to serve any content on netflix.com and tamper with authenticated

Netflix subscribers and their data. The tampering of subscribers and their data was enabled due to users' cookies being accessible from any subdomains.

1.1.2. Problem Statement

Research shows that architecture and design flaws are leading causes of vulnerabilities in software. According to the IEEE Computer Society's Center for Secure Design, while a system may always have implementation defects, security breaches in many systems are caused by design flaws [75]. During its ongoing security push, Microsoft reports more than 50% of the uncovered problems are architectural in nature [74]. According to the data published by MITRE, design weaknesses represent approximately 75% of the 25 most dangerous software errors and they also account for more than one-third of the current 940 known common weakness enumerations. Architecture and design flaws represent at least 50% of the total reported vulnerabilities in software systems [73].

There is a lack of consolidated design knowledge on how to build microservice applications. With an increase in the adoption of microservice architecture in the development of applications and the increase in security breaches in microservice based systems, there is a need to examine and understand security issues that exist in microservice architectures. Security issues can be summarized into five main points.

- Security is often considered as an afterthought, rather than during the early development phases, despite the increasing number of security breaches and incidents. This means that security aspects of the system are considered after the code has been written. Security considerations are thought of as roadblocks that prevent software from being released on time [8].
- There are more vulnerabilities per line of code in microservices than in equivalent monolithic applications. According to the Evolution of the Secure Software Lifecycle 2018 Application Security Statistics Report, it is reported that for every 100,000 lines of code,

there are 39 vulnerabilities in a traditional application. In comparison, in microservice architectures, there are 180 vulnerabilities [48].

- Microservices present new security challenges that are not present in monolithic applications due to the distributed nature of the architecture [18]. An example: communications between microservices are over the network which means a request may be susceptible to man-in the middle attacks.
- There is a lack of research in the area of microservice security [3, 23]. Microservice security is not very well understood in both industry and academia. There is a lack of comprehensive knowledge regarding how to build applications using microservice architectures with security in mind.
- IBM Market Development & Insights team [43] describes 53% of the respondents to their surveys considers security as one of the roadblocks in adopting or expanding the use of microservices in their company despites the advantages offered by microservice architectures. Pereira-Vale, A. et al. [3] and Berardi, D. et al. [23] also state that security is one of the main challenges in using microservice architectures to develop complex systems.

1.2.RESEARCH GOAL

The goal of the research is two-fold: 1) to study and document security properties that can remediate security issues in microservice architectures; and 2) define an effective approach to assist the software architects in formally defining security properties early on in the software development lifecycle. The research examines microservice security from the perspective of industry and academia. The questions driving the research are as follows:

RQ1: What are the security challenges in microservices architecture?

RQ2: What mechanisms are currently used to address the security challenges in microservices architecture?

RQ3: What approach can enhance the security modeling and specification in microservice architectures?

The expected outcome is a framework that provides sufficient support in formally defining security properties and adding structural elements (components and connectors) in the architecture that address software vulnerabilities in earlier stages of software development of microservice architectures. The intent is to provide a framework with clear guidelines on how to build applications using microservice architectures with security in mind. Such a framework would integrate security properties and support software architects regardless of their level of knowledge and experience in security.

1.3. Significance of the Research

This dissertation defines a framework to support the design of microservice architectures and remediate documented security issues. The framework enhances the ability of software architects to formally specify security properties early on in the software development lifecycle. It also includes the use of decision trees to guide software architects in determining what specific security properties should be considered, how different security properties are related and can be used together, and what additional structural elements (components and connectors) should be considered when adding specific security properties. These security properties are derived from existing security challenges and the corresponding security practices used to address them.

The impact of the work is that software vulnerabilities are addressed during early phases of software development (architecture and design) rather than later in the software development lifecycle. This helps to significantly reduce costs associated with software defect mitigation. Studies have shown that the cost ratio in tackling a software defect, including security vulnerabilities, is doubled if defects are discovered during the implementation phase compared to the architecture and design phases. This ratio more than triples if defects are discovered during testing. The work provides comprehensive support in defined security in microservice architectures, especially for software architects who have minimal experience in society.

1.4. Organization of Dissertation

This dissertation is organized as follows. Chapter 2 provides an overview of software architecture and microservice architecture. It also discusses the difference and similarities between microservice architecture and service-oriented architecture. The last section in chapter 2 provides an introduction to the Architecture Analysis & Design Language (AADL) and the existing security annex.

Chapter 3 presents the research that was done to create the framework that guides software architects in designing microservice architectures with security in mind. The chapter first presents the existing security challenges in microservice architectures. It is followed by descriptions of security practices that are used in the industry and described in literature. The chapter then presents the development of the framework.

Chapter 4 describes the experiment performed to evaluate the practicality of the framework and the observations and results of the experiment conducted in this dissertation. Chapter 5 describes related work on security analysis in software architecture and AADL security annex. Chapter 6 presents the summary of the work and discussion of future work.

Appendix A presents the decision trees. Appendix B presents the survey questions of the experiment. Appendix C presents the survey results from the experiment. Appendix D presents the research study background survey results.

Chapter 2: Background

This chapter is divided into four major sections. The first section provides a high-level overview of software architecture. The second section introduces microservice architecture. The third section describes the differences and similarities between microservice architecture and service-oriented architecture. The fourth section introduces architectural description languages, which are an important mechanism for formally defining security properties.

2.1. Overview of Software Architecture

Software architecture [9] describes the structure of a system, architecture characteristics the system supports, architecture decisions, and architecture principles governing the design and evolution over time. The structure of a system refers to the type of architecture style(s) the system is implemented in [9]. The architecture style describes the components of the system, behavior of each component, characteristics of the components (properties), and interrelationships among the components [34]. Architecture characteristics [9] are the “ilities” or “quality attributes”, such as availability, security, performance, and usability, that the system must support. They specify non-domain design considerations, influence structural aspects of the design, and are critical to the success of the application. Architecture decisions [9] are rules and constraints that govern how the system should be built and what development teams are allowed and not allowed to do, such as the presentation layer cannot access the database layer directly in a layered architecture. Architecture principles [9] are guidelines or preferred methods given a particular circumstance, such as asynchronous messaging between services can yield better performance in a microservice architecture, thus use asynchronous messaging whenever is possible.

When designing a system, software architects will first analyze the requirements in the problem domain to identify the architecture characteristics, such as performance, security, and availability, that the system needs to support. Based on the identified architecture characteristics and their priorities, architects will choose which software architecture styles would be suitable for the problem domain. Examples of software architecture styles are layered architecture, event-driven architecture, and microservice architecture [9].

In a layered architecture style [9], components are organized into layers with each layer responsible for performing a specific role, such as presentation, business, and database. Each layer provides an abstraction around the work that needs to be done to satisfy a particular business request and typically only accepts requests from the immediate layer above it. Layered architecture style is well suited for systems that require high testability and simplicity.

In an event-driven architecture style [9], components are decoupled, and they receive and process events asynchronously. There are two primary topologies with the event-driven architecture: the broker topology and the mediator topology. The broker topology is made up of four primary components: an initiating event, an event broker which contains at least one event channel, event processors, and processing events. The event flow begins with the initiating event being sent to an event channel in the event broker for processing. An event processor accepts the initiating event from the event broker and processes the event. Once the event processor completes the processing, it generates the next processing event and sends it to the event channel asynchronously for further processing. The other event processors listen for the next processing event and react to it accordingly. The process continues until no one is interested in what the final event processor did. The broker topology is great for systems that require extensibility, performance, responsiveness, and scalability. The mediator topology is made up of five components: an initiating event, an event queue, and event mediator, event channels, and event processors. The event flow begins with an initiating event being sent to the event queue. The event mediator, which is responsible for the workflow, accepts the initiating event from the event queue, creates the corresponding processing events, and sends them to specific event processors via dedicated event channels. The event processors process the processing events and provide responses to the event mediator. The mediator topology is great for systems that require recoverability, workflow control, and error handling.

2.2. Microservice Architecture

Microservice architecture has become a very popular architecture style in recent years [9], and according to a series of surveys conducted by IBM Market Development & Insights in 2021 [43], microservices are heavily used in many industries worldwide. Microservice architecture [65, 41, 23, 36] is a software architecture style where the software application is built as a composition of microservices with each microservice addressing a single business need. Each microservice runs in its own process, and is deployed independently of other microservices. Microservices communicate with each other via lightweight protocols, such as hypertext transfer protocol (HTTP).

Microservice is an independently releasable, deployable, technology agnostic, and business domain bounded and scoped component [8, 45]. The implementation details of a microservice are hidden. Data is typically isolated whenever possible. Coupling, including shared schemas and databases used as integration points should be avoided in microservices [9]. Services offered by the microservice are only exposed via network endpoints. Any changes made inside a microservice will not affect other microservices. Once the change is made, a microservice can be deployed and released without having to re-deploy other microservices. This makes each microservice independently releasable and deployable [8].

Each microservice can be implemented in any language and using any technology that best suits the purpose of the microservice and development experiences of the development team [11, 36]. Microservices communicate through lightweight messages via networks [11, 8]. This makes microservices technology agnostic.

Each microservice addresses a single business need. The size of each microservice should be relatively small due to the bounded context [8, 11, 36]. This makes each microservice business domain bounded and scoped.

2.3. Microservice Architecture vs. Service-Oriented Architecture

Microservice architecture has been seen as an evolution of service-oriented architecture (SOA). Richards [55] explains that while it is true that the two architecture styles share some characteristics, they have different taxonomy, service ownership model, service granularity, and sharing components. The shared characteristics are: a. Both architectures are distributed architectures where service components are remotely accessed through remote access protocol, such as representational state transfer (REST); b. Both architectures place an emphasis on services as their primary architecture components used to implement and perform functionalities.

Microservice architecture's taxonomy supports two types of services: functional services and infrastructure services. Functional services are business domain services and infrastructure services refer to nonfunctional tasks, such as, authentication, authorization, and monitoring. Functional services are accessible externally while infrastructure services are not exposed externally. SOA's taxonomy can have any number of service types; however, it typically supports the following four basic types: business services, enterprise services, application services, and infrastructure services. Business services are abstract and coarse-grain services that define the core business operations performed at the enterprise level. Enterprise services are concrete and coarse-grain services that implement the functionality defined by the business services. There is usually a middleware component that bridges the business services and enterprise services together. Application services are fine-grained and application specific services that are bound to a specific application context and provide specific business services that are not found at the enterprise level. Infrastructure services refers to nonfunctional tasks.

Services in microservice architecture are owned by application development teams, whereas services in SOA are owned by different business organizations. Services in SOA require coordination between different business organizations. It significantly increases the amount of effort and time required during development, testing, deploying, and maintaining the services.

Service granularity in microservice architecture is smaller than SOA. Services in microservice architecture are single-purpose services, whereas, the services in SOA can range in size from small application to large product or subsystem.

SOA is an architecture style that features "share as much as possible", whereas microservice architecture features "share as little as possible". SOA maximizes on component sharing, whereas, microservice architecture minimizes on sharing.

After reviewing the similarities and differences between microservice architecture and SOA, microservice architecture does share some of the same security challenges that exhibit in SOA. This research only focuses on security challenges and corresponding security mechanisms that are in microservice architecture regardless if they are also in SOA.

2.4. Architecture Analysis and Design Language (AADL)

Architecture description language (ADL) is a language used to describe a complex system at a high level of abstraction that exposes a system's structure as a collection of interacting components. It allows software engineers to reason about system properties, such as performance, schedulability, and security. There exists a number of ADLs, such as Aesop, Adage, Darwin, Rapide, SADL, UniCon, Wright, C2, Meta H, Acme, and AADL. The proposed work will be documented in AADL.

Architecture analysis and design language (AADL) [22] is a formal specification language that allows software engineers to define software, hardware, and physical system components, their interactions, and properties of the components. With the formal foundations and well-defined semantics, it provides software engineers the capability to perform different types of analysis, such as performance, security, and data integrity analysis, on the architecture based on standard properties. AADL is also extensible to support annotation of models with user defined and analysis-specific properties.

An AADL model is composed of component type and component implementation (please see Figure 1 and Figure 2). Component type represents the externally visible characteristics of a component, such as name, component category, interfaces, properties, modes, and logical flows. Component implementation represents a blueprint of its internal structure in terms of subcomponents. It defines subcomponents, connections, calls, and modes (if they are not defined in the component type), details the flows associated with the component type that traverse the various subcomponents, and adds or modifies properties that are previously defined in the component type.

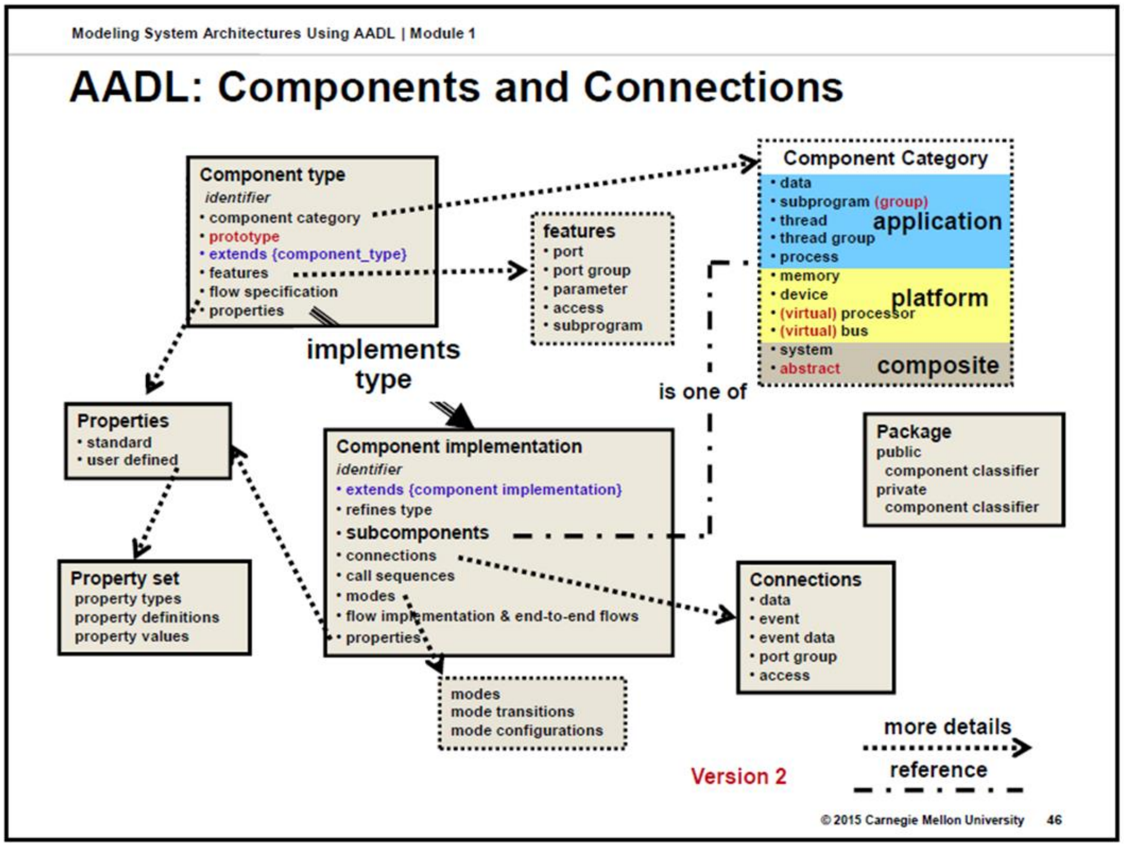


Figure 1: AADL Components and Connectors [80]

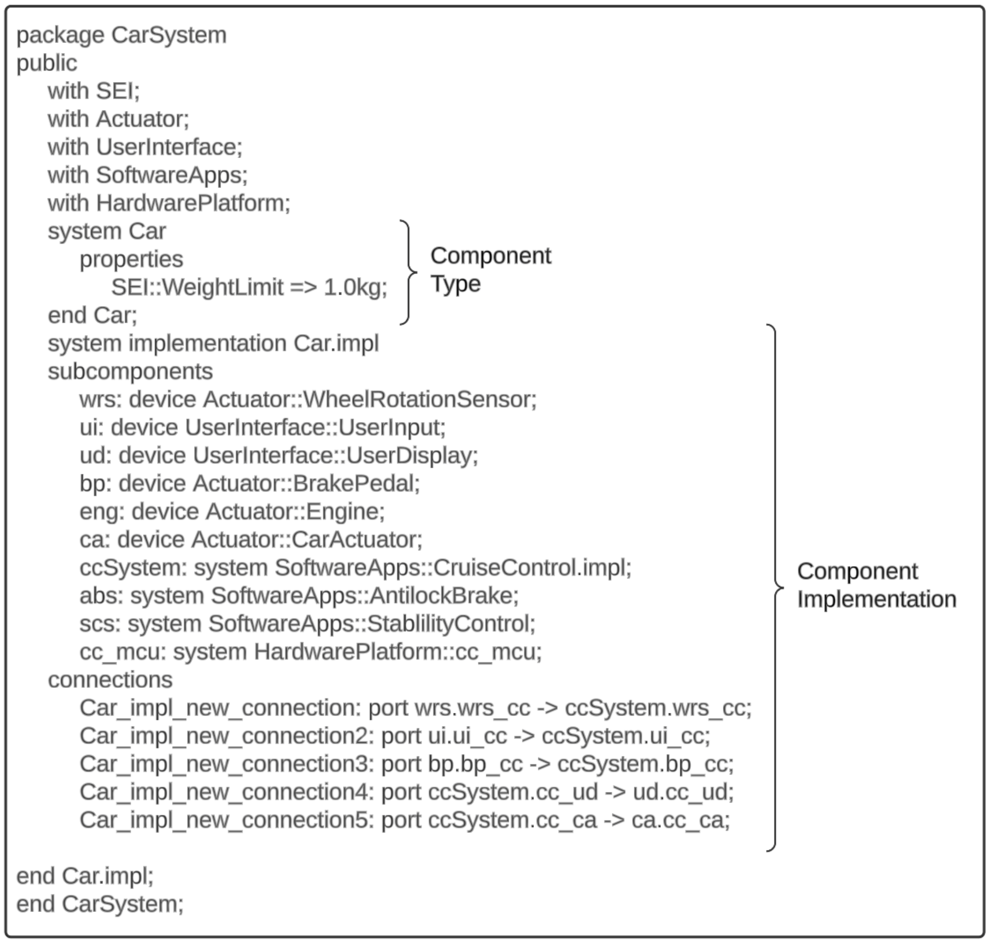
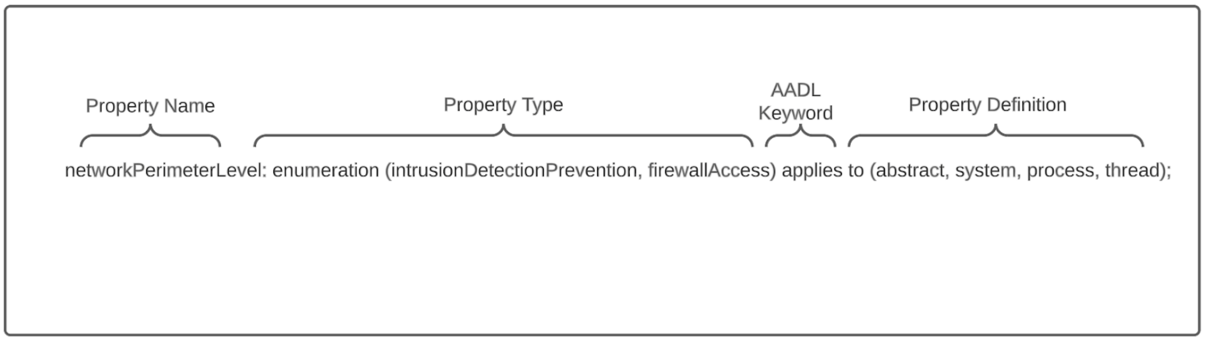

```
package CarSystem
public
    with SEI;
    with Actuator;
    with UserInterface;
    with SoftwareApps;
    with HardwarePlatform;
    system Car
        properties
            SEI::WeightLimit => 1.0kg;
    end Car;
    system implementation Car.impl
    subcomponents
        wrs: device Actuator::WheelRotationSensor;
        ui: device UserInterface::UserInput;
        ud: device UserInterface::UserDisplay;
        bp: device Actuator::BrakePedal;
        eng: device Actuator::Engine;
        ca: device Actuator::CarActuator;
        ccSystem: system SoftwareApps::CruiseControl.impl;
        abs: system SoftwareApps::AntilockBrake;
        scs: system SoftwareApps::StablilityControl;
        cc_mcu: system HardwarePlatform::cc_mcu;
    connections
        Car_impl_new_connection: port wrs.wrs_cc -> ccSystem.wrs_cc;
        Car_impl_new_connection2: port ui.ui_cc -> ccSystem.ui_cc;
        Car_impl_new_connection3: port bp.bp_cc -> ccSystem.bp_cc;
        Car_impl_new_connection4: port ccSystem.cc_ud -> ud.cc_ud;
        Car_impl_new_connection5: port ccSystem.cc_ca -> ca.cc_ca;

end Car.impl;
end CarSystem;
```

Figure 2: Component Type and Component Implementation

An AADL model is composed of the following elements:

1. Components
    a. Application software components: They refer to the applicative parts of the system. They include process, thread, data, and subprogram.
    b. Execution platform components: They refer to computing hardware and physical environment. They include processor, bus, memory, and device.
    c. System: It represents a composite of software and platform components or system components.
2. Connectors: They include port (data, event, and event data), subprogram, parameters, and subcomponent access.

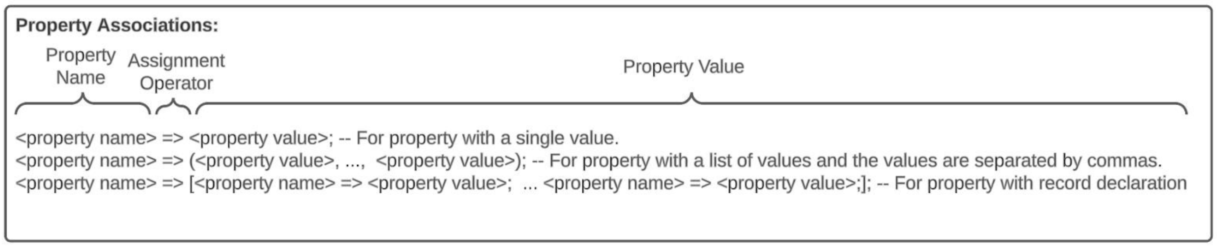
3. Properties: They define characteristics of the components and connectors. Each property has a name, a type, and definition (Please refer to Figure 3). The property type specifies the values that can be assigned to the property. The property definition specifies which AADL model elements the property applies to.



```
networkPerimeterLevel: enumeration (intrusionDetectionPrevention, firewallAccess) applies to (abstract, system, process, thread);
```

Figure 3: Structure of a Property

Property association is how values are assigned to properties and associated with AADL model elements (Please refer to Figure 4). A basic property association is made up of a property name, an assignment operator, a constant keyword (optional), and property value. Property value can be a single value whose type matches the type specified by the property, or a list of values separated by commas in parentheses of the same type if the property has been defined to accept a list. Operator +=> can be used to append value to a property that accepts a list of values. “In mode” keyword can be used to assign property values that only hold under certain conditions.

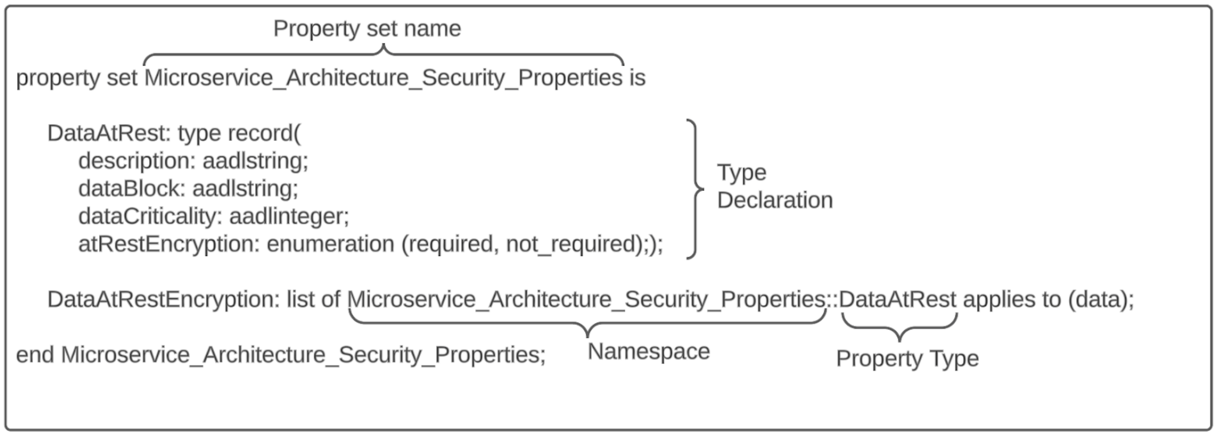


```
<property name> => <property value>; -- For property with a single value.
<property name> => (<property value>, ..., <property value>); -- For property with a list of values and the values are separated by commas.
<property name> => [<property name> => <property value>; ... <property name> => <property value>;]; -- For property with record declaration
```

Figure 4: Property Association

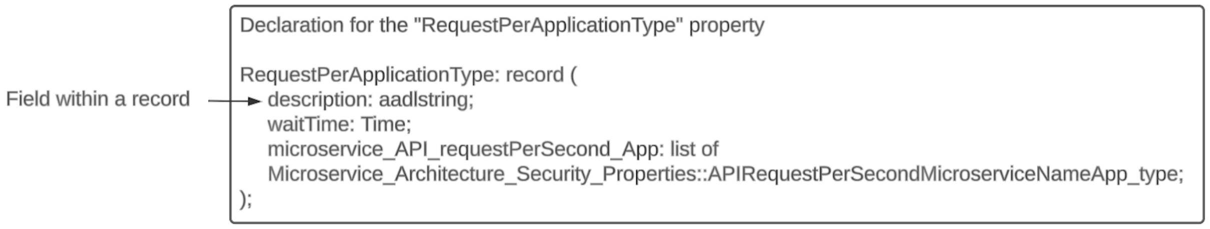
AADL supports two types of properties: AADL standard properties and user-defined properties. AADL standard properties are properties that are defined by the Software Engineering Institute and encompass common attributes for the AADL elements. User-defined properties are new characteristics to the AADL elements. The AADL annex describes a set of properties that augment the core AADL language with new elements.

AADL allows introduction of additional properties and property types through property sets. Each property set provides a separate name space (Please see Figure 5). A property in the property set can be referenced using "::" in the same or another property set. Property declaration defines a new property by declaring a property name and by specifying a property type. "Record" keyword is used when multiple fields are defined within a single structure (please refer to Figure 6). "Enumeration" keyword is used when a set of literal elements are defined (please refer to Figure 7).



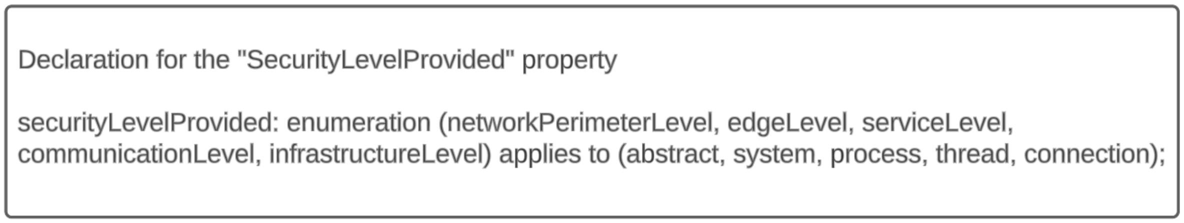
```
property set Microservice_Architecture_Security_Properties is

  DataAtRest: type record(
    description: aadlstring;
    dataBlock: aadlstring;
    dataCriticality: aadlinteger;
    atRestEncryption: enumeration (required, not_required););

  DataAtRestEncryption: list of Microservice_Architecture_Security_Properties::DataAtRest applies to (data);

end Microservice_Architecture_Security_Properties;
```

Figure 5: Property Set Example



```
Declaration for the "RequestPerApplicationType" property

RequestPerApplicationType: record (
    description: aadlstring;
    waitTime: Time;
    microservice_API_requestPerSecond_App: list of
    Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroserviceNameApp_type;
);
```

Figure 6: Property Declaration using Record



Declaration for the "SecurityLevelProvided" property

```
securityLevelProvided: enumeration (networkPerimeterLevel, edgeLevel, serviceLevel,
communicationLevel, infrastructureLevel) applies to (abstract, system, process, thread, connection);
```

Figure 7: Property Declaration using Enumeration

Chapter 3: Methodology

This chapter presents the work that was done to address the research questions:

- RQ1: What are the security challenges in microservices architecture?
- RQ2: What mechanisms are currently used to address the security challenges in microservices architecture?
- RQ3: What approach can enhance the security modeling and specification in microservice architectures?

3.1. Security Challenges and Security Practices in Microservice Architecture

This section addresses the research questions of the security challenges in microservice architectures (RQ1) and their corresponding security practices used and/or proposed to address the security challenges (RQ2). The security challenges described in this section are: authentication, authorization, logging, larger surface area, communication, patching, data, deployment, and trust.

3.1.1. Authentication

Authentication is the process of verifying the identity of an entity and checking who it claims to be [41]. The following two sections describe the challenges in authentication and the security practices used to address the challenges.

3.1.1.1. Challenges

In a microservice architecture, there are more authentication scenarios to consider compared to an equivalent monolithic architecture, such as authenticating end-users accessing microservices, authenticating microservices to other microservices, and authenticating external or third-party services connecting to microservices via API [13, 63, 36]. This increases the complexity in how authentication should be handled in microservice architectures.

Ayoub [13] and Fybish [63] state that since authentication is a cross cutting concern that affects every microservice in a given microservice architecture, some developers will create global authentication logic and assign the authentication responsibility to each microservice in the microservice architecture. Having each microservice responsible for more than its intended business needs is a violation of the single responsibility principle. Reusing the same code base creates a central code dependency and can negatively impact the technology agnostic aspect of microservices.

Management of credentials can be a challenge since there are significantly more credentials representing different user accounts, microservices, databases, and virtual machines compared to an equivalent monolithic architecture [63, 25, 8]. Pereira-Vale et al. [3] describes the challenge associated with maintenance and storage of authentication information. If authentication information is managed by an authentication microservice, an update is required whenever a new microservice or a new user is added. If the authentication information is managed by individual microservices, it increases the chances of the information being leaked should there be compromises happening to individual microservices.

3.1.1.2. Security Practices

API Gateway

Newman [8] suggests that API gateway can be used for coarse-grained authentication, such as preventing non-logged in users from accessing microservices, if the gateway can extract attributes about the principal as a result of the authentication. There are a number of issues associated with using API gateway for coarse-grained authentication. Coupling between microservices and API gateway will increase since the API gateway needs to know who is allowed access to which microservices. Since all traffic will have to funnel through the API gateway, the API gateway became the single point of failure. The more functionality the API gateway has, the greater the attack surface. Due to the mentioned issues, Newman [8] suggests the use of a gateway to handle security between users and services. The gateway can manage handshaking with an identity provider to perform authentication. Once the authentication is complete, the gateway passes information about the authenticated user to microservices with the assistance of Shibboleth or JSON web token.

Siriwardena and Dias [26] states that API gateway is the single-entry point to the microservice architecture and it provides the following security features: a. expose external-facing microservices via API to the client applications; b. work with an authorization server/service, such as OAuth, to secure APIs that are exposed at the edge; c. enforce only authenticated client applications with valid access tokens are allowed to communicate with microservices. It is done by extracting the access token from the request and introspecting it through the authorization server/service. If the access token is valid, it will forward the request from the client application to the appropriate microservice. The communication between client application and authorization server/service will affect the performance of the microservice architecture due to the increase in amount of communication. Alternatively, if the access token received by the gateway is a JSON web signature (JSON Web Token (JWT) signed by the authorization server), the gateway can verify the token by inspecting its signature. If the signature is from a trusted party, then the information contained in the token is trustworthy. One of the downsides of using JWT is that the gateway won't know if the token has been revoked since the gateway is not verifying the token with the authorization server. To address the revoking token issue, either the authorization server will inform the gateway that a token is being revoked or the token will be short-lived and require the token to be refreshed frequently. Another downside is that if the certificate used to verify the token is expired, the gateway will not be available to verify the signature of the token. Extra work is required to maintain the issuer's certificate. d. prevents throttling and DOS attacks.

Fybish [63] describes that API gateway can be used to control authentication for all downstream microservices. The advantage is that it is easy to implement. The disadvantages are: a. less secure because an attacker can gain access to any microservice once he/she bypasses the API gateway, b. increase in complexity and what the API gateway should be responsible for since the API gateway needs to manage different authentication rules for all microservices, c. API gateway becomes a single point of failure, and d. overheads in process and communication since different teams are responsible for the operations and maintenance of the API gateways.

Token-based Authentication

According to [70, 28, 26], tokens, such as API key and JSON web token (JWT), can be used for service-to-service authentication. API key is a simple bearer token that identifies a service and carries authentication information, such as ID and expiry time. It is generated by a developer from a developer portal, and it is sent with the request to a downstream (server) microservice. The downstream (server) microservice validates the identity of the upstream (client) microservice before processing the request.

JWT is a bearer token that carries claims about the service, expiry time, audience, and other standard JWT claims. JWT is either signed or encrypted by a symmetric authenticated encryption scheme. Validating of tokens will take place before a downstream (server) microservice accepts a request from an upstream (client) microservice.

JWT has the advantage over simple bearer tokens because it can be accepted by many different APIs due to the use of public key signatures. One of the disadvantages of using a bearer token is that it can be used by anyone, if captured, until it expires. Therefore, one must secure the communication channel with transport layer security (TLS) to reduce the risk of an intruder stealing the bearer token. The second disadvantage is that a portal for generating JWTs is required. The third disadvantage is that a mechanism needs to be in place to support revoking of tokens when a service retires. [70, 28, 26]

Madden [70] further suggests having the following four components when using token-based authentication: client-side token storage, server-side token storage, hash-based message authentication code (HMAC) token store, and a standard way to communicate tokens between client and server. Since authentication tokens need to be validated on every request, it is important to consider that when selecting which database to use to store the tokens since the database transaction for every lookup can be costly. The recommendation is to use non-relational database backends, such as Redis in-memory key-value store or NoSQL JSON. To lower the risk of various threats, such as tokens being injected to the database, tokens being modified, tokens being deleted, and tokens being stolen and replayed to API, the tokens in the database should be stored using hash-based message authentication code (HMAC) to protect tokens against tampering and forgery, the database should be separated from the API server to ensure that external clients do not have direct access to the database, and communication between the database and the API server should be protected with TLS. Each of the components should be classified with a different trust boundary. The other aspect to consider when dealing with tokens is token deletion. Once a token is deleted, it should never come back to life.

Subramanian and Raj [57] describes how token-based authentication works. The client application makes a request to an authentication server for an access token. After the authentication server validates the mandated credentials from the client application, it issues an access token to the client application. The client application sends the access token in the Authorization HTTP header with an API request. The API gateway validates the access token with the authentication server. Once the access token is validated, the API gateway forwards the request along with the access token to the corresponding microservice.

According to [26], JWT can be used in external to service authentication. An external application requests an access token from the security token service (STS). The STS generates a JWT which contains user context related to the external application. The external application can then use the JWT in an HTTP header when invoking a microservice.

Yarygina and Bagge [18] describes the use of security tokens such as JWT for user to service authentication. Once the user is authenticated by an authentication service within the microservice architecture, a security token is generated to represent the client’s identity. The security token will be sent to the client via TLS. The client will provide the security token whenever it makes a request. The security token along with the request will be passed from one microservice to another microservice to complete the request. Upon receiving the security token, a microservice will validate and verify the security token before processing the request. If the security token is invalid, the microservice will reject it and stop the request processing.

Certificate-based Authentication

Mateus-Coelho et al. [4], Siriwardena and Dias [26], Yarygina and Bagge [18], Newman [8], and Barabanov and Makrushin [28] suggest the use of certificates along with TLS/MTLS for service authentication.

Certificate-based authentication [35, 14] is a cryptography technique that uses a certificate to identify an entity before granting any type of access [60]. A certificate contains information about its owner, a public key of the owner, and information about its issuer. The following explains how the certificate-based authentication works in a microservice architecture along with TLS/MTLS communication protocol. A microservice needs to request a certificate from a certificate authority (CA). It is done by submitting a certificate signing request (CSR) form along with its public key to the CA. Once the CA completes the verification process of the information on the CSR, it will sign it and send it back to the microservice. Before communication between downstream (server) and upstream (client) microservices can take place, they need to authenticate themselves to each other. When the TLS communication protocol is used, the downstream (server) microservice will provide its signed certificate along with its public key to the upstream (client) microservice. The upstream (client) microservice will verify the downstream (server)'s certificate by checking the signature of the CA who signed it. If the CA is trusted and the downstream (server) microservice's certificate is valid, the authentication process is complete. The upstream (client) microservice creates a session key that will be encrypted with the downstream (server) microservice's public key. The downstream (server) microservice decrypts the session key, generates an encrypted acknowledgement with the session key, and initiates encrypted communication with the upstream (client) microservice. When the MTLS communication protocol is used, an upstream (client) microservice needs to authenticate the downstream (server) microservice and vice versa. Once the microservices are done verifying each other's authenticity, then they can communicate with each other after exchanging the session key.

An advantage in using certificate-based authentication is that the certificates MTLS uses are time-bound. In the event that the certificate and the corresponding private key are compromised, the vulnerability is limited by the lifetime of the certificates. A disadvantage is that certificate-based authentication is a centralized solution that is not very scalable and the implementation can be complex [25].

API Key-based Authentication

Mateus-Coelho et al. [4] and Newman [8] suggest the use of API keys for service-to-service authentication. With API keys, the downstream (server) microservice generates a unique key for each of the upstream (client) microservices. Whenever an upstream (client) microservice makes a request to the downstream (server) microservice, the upstream (client) microservice sends the request along with a unique key. The downstream (server) microservice verifies the upstream (client) microservice's key. If the key is valid, the downstream (server) microservice processes the request. If the key is not valid, the downstream (server) microservice rejects the request.

API key-based authentication is easy to implement and use compared to other types of authentication methods. The authentication is done by including a key in the request and verifying the key is valid. Most developers are familiar with API keys, and therefore, no extra training is required. However, if the key is stolen, any microservices will be able to use it to request all the services associated with the key from the downstream (server) microservice as if they are the owner of the key. If any of the services involve write, update, or delete access to any data, it can be a huge security concern. It is not very secure compared to other technology and it can be leaked easily, such as showing up in logs or extracting the API keys from code [49]. To reduce the security concern, it is recommended to limit API Key-based authentication for services that involve read-only data [44].

Hash-based Message Authentication Code

Mateus-Coelho et al. [4] suggests the use of hash-based message authentication code (HMAC) for service-to-service authentication. HMAC is a cryptographic technique that uses a hash function and a secret key for authentication [31]. HMAC [4, 31] works as follows: The downstream (server) and upstream (client) microservices share a secret key and have a mutual agreement on how a message digest is calculated. When the upstream (client) microservice wants to send a request to the downstream (server) microservice, the upstream (client) microservice first creates a message digest by combining the request and the secret key, and then sends the message digest along with the original request to the downstream (server) microservice. Upon receiving the message digest and the original request, the downstream (server) microservice calculates a message digest by hashing the original request with the secret key that it shares with the upstream (client) microservice. If the calculated value matches the digest sent by the upstream (client) microservice, then the data integrity and authenticity of the request is guaranteed. The integrity of the request is preserved because the message was not modified in transit. The authenticity of the microservice is known because the downstream (server) microservice knows who the upstream (client) microservice is.

OpenID Connect

Góes de Almeida et al. [41], Banati et al. [25], Mateus-Coelho [4], and Yarygina and Bagge [18] mention OpenID Connect (OIDC) as one of the authentication protocols used on top of OAuth 2.0 used for user authentication in a microservice architecture.

OIDC [42] is a protocol that provides an identity service layer that sits on top of OAuth 2.0. It allows the delegation of the responsibilities of user authentication and claim generation of authenticated users and authentication events to authorization servers.

The following explains how OIDC can be used in a microservice architecture for authentication. It begins when a user (end user) wants to access a microservice (relying party). The microservice redirects the user to an OpenID provider, which is an authorization server that has implemented OIDC. An OpenID provider is used to authenticate a user and return claims about the authenticated user and authentication event. The user interacts with the OpenID provider to get authenticated. If the authentication is successful, the user is being redirected back to the microservice with the authorization code. From the OpenID provider, the microservice can obtain an ID token, access token, and optionally a refresh token with the authorization code. If the microservice needs additional information about the user, it can use the access token at the OpenID provider's userinfo endpoint. With the ID token, the microservice has proof that the user has been authenticated [42]. The microservice can use the ID token to communicate with other microservices on the user's behalf. The other microservices can validate the signature on the ID token with the public key of the OpenID provider before providing a response.

An ID token is a security token which contains claims about an authenticated user and event. It is used to convey claims to a microservice about an authenticated user and event, and it is encoded in JSON Web Token (JWT) format. The ID token has three parts: header, payload, and signature. The header contains metadata of the token, such as, the type, and the signature algorithm used to protect the integrity of the claims in the payload. The payload contains claims about the authenticated user and event. The signature contains a digital signature created based on the payload and the secret key of the OpenID provider. A microservice can validate the signature on the ID token with the public key of the OpenID provider. It calculates the hash of the payload, decrypts the digital signature with the public key of the OpenID provider, and compares the hashes. If they match, then the integrity of the claims in the payload is preserved. [42]

Federated Identity Solution

Rountree [38] and [68] describe the use of federated identity solution to separate user authentication from application logic and to delegate authentication to an identity provider. The federated identity solution is composed of two required components: identity provider and service provider. The identity provider is responsible for authenticating entities against its credential store. Once authentication is complete, the identity provider will allow access to the user's identity information. The service provider is responsible for providing services to others based on the user's identity from the identity provider. It trusts the user's identity from the identity provider and will not perform additional authentication.

The following explains how federated identity solution can be used in a microservice architecture for authentication. When a client application wants to access a microservice, the client application needs to be authenticated by an identity provider. The identity provider authenticates the client application against its credential store. If the authentication is successful, the identity provider will issue a security token that contains claims about the user's identity. A security token service might transform and augment claims in the token issued by the identity provider, when necessary, before the security token is sent to the client application [68]. The client application can then use the security token to request service from the service provider. The service provider trusts the claims in the security token and will not perform additional authentication.

The advantages of using federated identity solution are [38, 68]:

- Identity provider is the only component that has access to user's credentials. Microservice only has access to the user's identity information provided by the identity provide and not the user's credentials. In the event that a microservice is compromised, no user's credentials are exposed.
- Authentication is separated from the microservice business logic. It simplifies the development of microservice.

The disadvantages of using federated identity solution are [38, 68]:

- If the identity provider is compromised or the credential the client application uses to log in is compromised, an attacker can gain access to all the microservices the user credential has access to.
- The use of federated identity solution requires infrastructure setup, support of extra hardware and software, and conformation to the standards followed by other organizations. The cost of using federated identity solution might outweigh its benefits.
- Authentication can be a single point of failure.

3.1.2. Authorization

Authorization is the process of granting an entity permission to do or own something [79]. An entity can be a person or a system. Each entity should only be able to perform actions on microservices it is allowed to. The following two sections describe the challenges in authorization and the security practices used to address the challenges.

3.1.2.1. Challenges

Managing credentials and their access rights in a microservice architecture is more challenging since there are a lot more credentials representing different user accounts, microservices, databases, virtual machines, and other components in a microservice architecture compared to an equivalent monolithic architecture [8]. There are more authorization scenarios to consider in a microservice architecture compared to an equivalent monolithic architecture, such as authorizing a microservice to call an API on the user's behalf and authorizing microservices to access other microservices [63, 25]. This increases the complexity in how authorization should be handled in microservice architectures.

Ayoub [13] and Fybish [63] state that since authorization is a cross cutting concern that affects every microservice in a given microservice architecture, some developers will create global authorization logic and assign the authorization responsibility to each microservice in the microservice architecture. Having each microservice responsible for more than its intended business needs is a violation of the single responsibility principle. Reusing the same code base creates a central code dependency and can negatively impact the technology agnostic aspect of microservices. Banati et al. [25] states that if a microservice is required to handle authorization at the service level and needs to store and administer user's data, it increases the chances of personal information being leaked and accessed by unauthorized entities.

When it comes to the container-based microservices, maintaining service credentials and access control policies can be more challenging. According to [26], a container-based microservice is immutable meaning that once the container is up, it does not maintain any runtime states or any changes made to its file system. It means that extra steps need to be taken to maintain the dynamic list of allowed clients and access control policies and service credentials since service credentials would be rotated periodically.

Newmon [8] describes the confused deputy problem as one of the authorization challenges. The confused deputy problem refers to an upstream (client) microservice tricking downstream (server) microservices into doing something they should not be doing.

3.1.2.2. Security Practices

API Gateway

API gateway can be used to centralize the enforcement of coarse-grain authorization at the edge for all downstream (server) microservices [28, 63, 26]. That way, each microservice does not have to worry about access control to its services.

The disadvantages of using API gateway to perform authorization are [20, 63]:

- When the microservice architecture grows, the authorization decisions can get very complicated. If all authorization decisions are put on the API gateway, the API gateway can become unmanageable.
- API gateway becomes the single point of failure.
- API gateway is typically owned by the operation teams. API is owned by development teams. Development teams need to communicate with the operation teams whenever an authorization rule requires changing. It increases the overheads in process and communication.
- The use of API gateway can make the microservice architecture less secure when an attacker bypasses the API gateway and gains direct access to microservices.

The advantage of using API gateway to perform authorization is that it is easy to implement since it is centralized [63].

Security Token

Yarygina and Bagge [18] suggests the use of security tokens along with access control mechanisms for user authorization. Security tokens can carry authorization information of the user. Based on the authorization information, the system can determine which microservice it can request service from.

Banati et al. [25] describes the use of tokens for user authorization. After the user has been authenticated, the identity and access management component generates a time-sensitive JSON Web Token (JWT). The JWT is then appended to every request. Microservices are able to manage the users and their rights based on the information specified in the token.

OAuth 2.0

OAuth 2.0 [16] is an authorization framework that gives applications a way to make API requests without the need for users to share their credentials and with limitations on what the applications can do. Microservices do not have to worry about a user's credential when a microservice is trying to access features from another microservice on the user’s behalf. Góes de Almeida et al. [41], Banati, A. et al. [25], and Yarygina and Bagge [18] suggest the use of OAuth 2.0 for authorization in a microservice architecture.

The following explains how OAuth with proof key for code exchange (PKCE) can be used in a microservice architecture for user to service authorization involving client applications, e.g., desktop applications, web applications, or mobile applications, that are capable of handling HTTP redirects. If the client application is not registered with the OAuth server, it needs to complete the registration first to obtain a pair of credentials, which includes a client application identifier and client application secret. The application secret is only used when a client application is capable of keeping a secret. Some client applications, such as JavaScript applications, cannot keep a secret, therefore, the application secret is not used. [16]

It begins with a user (resource owner) who wants to use an application to access a microservice (resource server) on his/her behalf. The application generates a PKCE secret (code verifier) and hashes it. The generation and hashing of PKCE secret and is done on every request. The application redirects the user's browser to an OAuth server's authorization endpoint along with the hash (code challenge). Once the user (resource owner) completes the authentication, the OAuth server (authorization service) requests confirmation from the user regarding the application's request to access the microservice. Once the user confirms, the OAuth server generates and sends an authorization code to the user's browser. The user's browser then sends the authorization code to the application since there is no direct communication between the OAuth server and the application. The application makes a POST request with the authorization code, the application ID, the application password, and the plaintext PKCE secret to the OAuth server's token endpoint in exchange for an access token. The purpose of sending the plaintext PKCE secret is to confirm that the sender of this POST request is really the application itself. The OAuth server calculates the hash using the PKCE secret and compares it with the hash sent by the user previously. If they match, then the OAuth server knows it is the application who is making the request for the access token and sends the application the access token. The application can then communicate with the microservice to access the resource with an access token. The microservice verifies the application's access token with the OAuth server before allowing access to the requested resource [16]. The access token is meant for resource access and is not intended to convey information about the authentication event or the user. The authentication step in OAuth is used to validate a user's entitlement to give consent to authorize an access request for a resource. [42].

If an application is an Internet of Things (IoT) device, such as an Apple TV, the device authorization flow is used instead. The flow begins when the user wants to use a primary device that requires access to a microservice. The primary device must be able to make outbound HTTPS requests. The primary device sends an authorization request along with the device ID (client ID) to the OAuth server. The OAuth server sends a device code, end-user code, and a user verification universal resource identifier (URI) to the primary device. The primary device shares the end-user code and verification URI with the user. The user accesses the verification URI on a secondary device. The secondary device must be capable of supporting user interaction to authenticate and authorize API requests to the microservice. Once the user is authenticated, the OAuth server asks for the user for the user code and for authorization of the API call. The user provides the user code and consent. The primary device continues to poll the OAuth server. Once the user consents, the OAuth server responds to the primary device's next polling request with an access token and a refresh token if applicable. The primary device can use the access token to call the microservice's API on the user's behalf. [42]

The communication involving the user's browser is considered as the front channel. Front channel passes data via the browser's address bar as a redirect which is susceptible to request and response being modified by malicious parties. The communication between the application and OAuth server is considered as the back channel. Back channel involves HTTPS requests to and from application to server, the communication channel is encrypted and cannot be tampered with. Refresh token is a special token used to get a new access token without having the user visit the OAuth server and to keep the user logged in. The refresh token is always between application and authorization server. API does not accept refresh tokens [16].

The use of OAuth in a microservice architecture provides consistency in user experience as far as how authorization is handled and how security is managed. This makes it easier for users to identify fake authorization prompts [16]. The drawback in using OAuth is performance due to the increased communications between microservices and OAuth server.

Certificates

Yarygina and Bagge [18] suggests the use of certificates for authorization. If the microservice architecture uses MTLS with a self-hosted public key infrastructure, a certificate should be created for each microservice type. To allow a microservice to have access to another microservice, a trust list by certificate type should be established. By default, no microservice will allow access to another microservice. If one microservice needs to have access to another microservice, the microservice needs to be added to another microservice's trust list prior.

Access Control System

Barabanov et al. [28] describes three different ways in how an access control system can be used for service level authorization in a microservice. In a typical access control system, the following components are included: a. Policy administration point (PAP) which allows an administrator to define and maintain access control rules via an user interface; b. Policy decision point (PDP) which uses access control rules defined in PAP to make access decisions; c. Policy enforcement point (PEP) which enforces the access decisions made by the PDP in response to incoming requests; and d. Policy information point (PIP) which maintains additional attributes that can assist PDP when making access decisions. The three different ways are a. Decentralized pattern, b. Centralized pattern with a PDP, and c. Centralized pattern with an embedded PDP.

When an access control system is implemented using the decentralized pattern, each microservice is responsible for making access decisions (PDP) and enforcing the access decisions made by the PDP (PEP). This pattern offers more fine-grained access control because the access control rules are more domain specific. However, the development team must be able to configure the access control rules correctly and manual configuration is not scalable.

When an access control system is implemented using the centralized pattern with a PDP, each microservice is responsible for enforcing access control decisions (PEP). The defining of access control rules (PAP), the decision making based on access control rules (PDP), and the maintenance of additional attributes (PIP) are shared among all microservices in the same architecture. This pattern offers flexibility in managing the access control rules, access decision policies, and attribute collection since they are decoupled from the microservices who use them. However, the latency suffers due to additional network calls from the microservices to the PDP. It is recommended to implement this pattern along with the other patterns to avoid single point of failure and to enforce defense in depth principle.

When an access control system is implemented using the centralized pattern with an embedded PDP, each microservice is responsible for making access decisions (PDP) and enforcing the access decisions made by the PDP (PEP). The access control rules (PAP) and attributes (PIP) are defined centrally and are delivered to embedded PDP in the microservice. Latency is not affected by this pattern due to the embedded PDP. It is recommended to implement this pattern along with the other patterns to avoid single point of failure and to enforce defense in depth principle and to beware of the approach used to propagate the update from the centralized PAP to each microservice.

Centralized Upstream Authorization and Decentralized Authorization

Newman [8] suggests two mechanisms in addressing the confused deputy problem: a) centralized upstream authorization and b) decentralized authorization. Centralized upstream authorization refers to all required authorization to be performed as soon as the request is received in a system, and once all required authorization is processed, the downstream microservices can assume the requests are allowed under the implicit trust principle. The issue with the centralized upstream authorization is that the upstream microservice or gateway has knowledge of the functionality provided by the downstream microservices and the access control of those functionalities. This violates the principle of independent deployability and creates the single point of failure.

Decentralized authorization refers to having the downstream microservice where the functionality being requested lives to handle the authorization based on information of the requestor. The issue with decentralized authorization is that microservice has additional functionality on top of the single business need it is responsible for. The other issue is that additional information needed to process authorization needs to be passed from one upstream microservice to downstream microservice.

3.1.3. Logging

3.1.3.1. Challenges

When microservices are spread across different platforms, security may be out of the control of the microservices owners and completely dependent on the platform environment owner. Collecting the required and necessary information to diagnose what went wrong and correlating requests among microservices become challenging [4, 26, 8, 32, 19]. For microservices that are deployed using containers, the audit logs are not kept at each node running the microservices [26].

3.1.3.2. Security Practices

The use of distributed tracing systems, such as Jaeger and Zipkin, and logs to keep track of essential information that can provide knowledge about exploitation, how the system was used, and weak points are proposed [26, 4].

Since it is common for microservices to be built with different technologies, it makes the structure of the logs and the amount of information collected even more important. The structure of the logs will impact how they need to be parsed and how logs can be combined to represent complete requests for analysis. The amount of information collected will also impact the level of difficulty in diagnosing a problem. It is recommended to collect the following information as a minimum: name of the service, name of the logged-in user, IP address, correlation ID, time at which the message arrived, time taken, name of the method, call stack, and HTTP code [12].

3.1.4. Communications

3.1.4.1. Challenges

Sun et al. [32] and Henrique et al. [19] describe communication between microservices as one of the security challenges in a microservice-based system. In a microservice-based system, microservices are required to communicate with each other over the network in order to complete requests. If the communication between microservices is not secured, it will expose the microservice-based system to different types of attacks, such as man-in-the-middle attack and session/token hijacking [2, 4]. On top of that, microservices can be developed by different teams. Improper interception and inappropriate access can happen if the teams do not agree on the communication protocol between microservices.

3.1.4.2. Security Practices

TLS

Siriwardena and Dias [26] and Yarygina and Bagge [18] describe the use of transport layer security (TLS) to encrypt network traffic between microservices and to protect communication between microservices for confidentiality and integrity. TLS can be used by any application-layer protocol to secure communications, such as Java Database Connectivity over TLS and Simple Mail Transfer Protocol over TLS. TLS also provides one way authentication where the downstream (server) microservice provides a certificate to the upstream (client) microservice for identity verification before the microservices communicate with each other.

To enable TLS communication, the key provisioning process steps are followed: 1. A private and public key pair is generated for each microservice, 2. A certificate-signing request is generated and submitted for approval to the team who owns a corporate certificate authority (CA), 3. A CA-signed certificate is generated for each microservice, and 4. The key pair and certificate are deployed with each microservice. The key provisioning process can be done manually or facilitated by a certificate management framework, such as Lemur.

MTLS

In addition to TLS, Siriwardena and Dias [26] and Yarygina and Bagge [18] also suggests the use of mutual transport layer security (MTLS) to encrypt network traffic between microservices and to protect communication between microservices for confidentiality and integrity. MTLS also provides two-way authentication where the downstream (server) microservice provides a certificate to the upstream (client) microservice for identity verification and the upstream (client) microservice provides the downstream (server) microservice for identity verification.

3.1.5. Data

3.1.5.1. Challenges

In a monolithic architecture, data is typically stored in a centralized database and accessed by modules within the architecture when needed. In a microservice architecture, data is typically owned and stored in each microservice. It is not a requirement that each microservice must own and store data. To fulfill a request in a microservice architecture, it is very common that multiple data sets are accessed in various microservices. Comparatively speaking, data moves around an architecture more often in a microservice architecture than in a monolithic architecture, and this makes securing data more challenging [8].

Data in Transit

Newman [8] describes four main challenges regarding data in transit. The first and second challenges are about the identity of downstream (server) and upstream (client) microservices. Downstream (server) microservice refers to the microservice receiving a call from another microservice. Upstream (client) microservice refers to the microservice making a call to another microservice. Newman [8] suggests the need for the upstream (client) microservice to verify the identity of the downstream (server) microservice to ensure that the upstream (client) microservice is communicating with an authentic microservice. A malicious party can impersonate a downstream (server) microservice in an attempt to steal all the receiving data. The downstream (server) microservice needs to verify the identity of the upstream (client) microservice to ensure it is an authentic microservice requesting for service. A malicious party can impersonate an upstream (client) microservice in an attempt to request for data that it does not have access to. The third challenge is about the visibility of data. When data is sent across the network between an upstream (client) and a downstream (server) microservices and vice versa, it is possible for a malicious party to see the data. The fourth challenge is about data manipulation. When data is sent across the network between an upstream (client) and a downstream (server) microservices and vice versa, it is possible for a malicious party to manipulate the data.

Data at Rest

Many of the high-profile security breaches involve attackers acquiring and reading data at rest. Newman [8] states that one of the root causes of security breaches is because data is stored in an unencrypted form. Once a malicious adversary is able to compromise the microservice, he/she can have unlimited access to the data stored within the microservice. Another root cause is that there are fundamental flaws with the protective mechanism used on data.

Data Sharing

In a monolithic application, data is shared via session or can be accessed via a centralized database. In a microservice architecture, data is stored and owned by each microservice. When a downstream (server) microservice needs data about a request, the upstream (client) microservice needs to pass the requested data explicitly to the downstream (server) microservice. It is possible that a malicious adversary can modify the data during transit [26].

There is also a question of how much data should be sent across the network in order to fulfill a request and how much data does each microservice require in performing its part of the request since it is very rare for a request to be fulfilled by a single microservice in a microservice architecture.

3.1.5.2. Security Practices

TLS and MTLS

Newman [8] suggests the use of transport layer security (TLS) and mutual transport layer security (MTLS) to protect data in transit. TLS provides encryption of data which prevents data from being visible to other unauthorized entities when data is being sent between two microservices. It also provides authentication of the downstream (server) microservice when two microservices are communicating with each other. The downstream (server) microservice provides a certificate to the upstream (client) microservice. The upstream (client) microservice verifies the identity of the downstream (server) microservice before establishing communication between microservices. TLS is recommended when HTTP communication is used. If both the downstream (server) and upstream (client) microservices require authentication, then MTLS should be used to protect data in transit. With MTLS, on top of the downstream (server) microservice providing a certificate to the upstream (client) microservice for verification, the upstream (client) microservice needs to provide a certificate to the downstream (server) microservice for verification before establishing communication between microservices.

MTLS and Service Mesh

When it comes to the identity of the upstream (client) microservice, Newman [8] suggests a number of ways to address the issue. The downstream (server) microservice can request the upstream (client) microservice to provide additional information, such as client-side certificate and a shared secret, to prove who it is. MTLS via certificates and service mesh can also be used to handle the authentications between upstream (client) and downstream (server) microservices.

Secure Communication Protocols

Newman [8] suggests the use of secure communication protocols, such as HTTPS, and message authentication code, such as hash-based message authentication code, to guarantee the integrity of the data from the upstream (client) microservice.

Encryption

For protecting the data at rest, data should be encrypted by well-known encryption algorithms [8, 4, 26]. Encryption can be done at the disk-level and application-level [26]. However, Newman [8] points out while it is good to encrypt everything, the downside is computational overhead. He suggests breaking down microservices into more fine-grained microservices, evaluating which data set is critical to the operation and requires storing, and identifying which data set contains sensitive information that requires encryption.

JWT

According to [26, 8, 28, 18], JSON Web Token (JWT) can be used to secure data in transit. The JWT is a container that carries contextual data and is passed from one microservice to another microservice so that microservices can share context. Upon receiving the JWT, a microservice validates the signature of the JWT before processing the request. Some microservices will also validate the audience field of the JWT. If the JWT is invalid, the microservice will reject it and stop the request processing.

Yargina and Bagge [18] and Barabanov and Makrushin [28] also suggest the use of security tokens such as JWT for propagating user identity throughout the microservices architecture. Once the user is authenticated by an authentication service within the microservice architecture, a security token is generated to represent the client's identity. The security token will be sent to the client via TLS. The client will provide the security token whenever it makes a request. The security token along with the request will be passed from one microservice to another microservice to complete the request. Upon receiving the security token, a microservice will validate and verify the security token before processing the request. If the security token is invalid, the microservice will reject it and stop the request processing.

There are two types of JWTs. The first type of JWTs is issued by a security token service (STS) that is trusted by all microservices in the same trusted domain governed by the STS. It is typically used when authentication is not required between microservices. Depending on the application scenario and the level of trust in the microservice deployment, microservice might request a custom JWT to be generated by the STS for each service interaction.

The second type of JWTs is self-issued by an individual microservice using its own private key. The self-issued JWT is passed as an HTTP header along with the request to the downstream (server) microservice over TLS. The choice of TLS as the communication protocol is to protect the confidentiality and integrity of the communication since JWT is a bearer token and to minimize the risk of an attacker stealing the token. The downstream (server) microservice verifies the JWT using the upstream (client) microservice's public key. Self-issued JWT also offers nonrepudiation since the contextual data is bound to the upstream (client) microservice. Secure data in transit is achieved since microservices cannot modify the content carried by the JWTs, and different types of validations, e.g., signature and audience, are performed before accepting JWTs. Siriwardena and Dias [26] suggest that self-issued JWT for each service interaction is generally more secure than using a shared JWT because the JWT will have a specific audience. The need to have a JWT for each service interaction depends on the level of trust in the microservices deployment.

Newman [8] mentions three issues to watch out for when it comes to using JWT tokens. The first issue is about key management. In order for a downstream (server) microservice to verify a signed JWT token, it needs to have access to the public key of the upstream (client) microservice. The downstream (server) microservice needs to know where to find the public key of the upstream (client) microservice. The maintenance of the lifecycle of public keys can become an issue. The downstream (server) microservice needs to figure out when and how often the public key of the upstream (client) microservice would change. The second issue is about expiration of tokens. Architects need to understand the impact of long expiration time on tokens and the security of the system. Some processes are asynchronous and might take a long time to complete their tasks, and hence the long expiration time on the token. The last issue is the amount of information each token should hold. Architects need to understand the impact of holding too much information in each token and the security of the system.

3.1.6. Patching

3.1.6.1. Challenges

In July 2017, Equifax [52, 61] disclosed a data breach resulting in personal identifiable information of at least 145.5 million individuals in the U.S. and nearly 1 million people outside the U.S. being accessed and/or stolen by attackers. The root cause of the breach was caused by a known vulnerability in Apache Struts Web Framework that was not patched within Equifax's infrastructure. The vulnerability allowed attackers to execute commands on affected systems.

The United States Computer Emergency Readiness Team publicly announced this vulnerability two days prior to the attack taking place. Apache Software Foundation released a patch for the vulnerabilities on March 7th, 2017. Equifax administrators were instructed to apply the patch to any vulnerable systems on March 9th, 2017, however, the patch was not applied to any of their vulnerable systems. On March 15th, 2017, the scans performed by Equifax did not flag any of the vulnerable systems. During the same month, Mandiant, a security consulting firm, was hired to investigate a series of incidents where criminals used stolen social security numbers to log into Equifax sites. They issued warnings to Equifax about multiple unpatched and misconfigured systems.

Attackers were able to take advantage of the vulnerability in Apache Struts Web Framework and gained unauthorized access to Equifax's online dispute portal. In subsequent months, they were able to access other systems and retrieve personal identifiable information and unencrypted usernames and passwords. The retrieved usernames and passwords were used to access additional 48 databases. The attackers disguised the data as normal network traffic so that they were able to remove the data without being detected.

Equifax's data breach incident highlights the importance of patching and the potential consequences of failing to keep up with patching in one's infrastructure. Equifax's data breach is not an isolated incident. There are many more attacks in which failure to keep up with patches is the leading cause. As companies continue to develop and deploy complex systems, the challenges of keeping up with patching will increase [8].

Newman [8] describes another challenge with patching that involves the infrastructure and software that the microservices run on. It is important to know who owns the infrastructure and software that the microservices run on. The ownership will impact who is responsible for maintaining and patching the infrastructure and software that microservices run on and how often they will be patched. Venčkauskas et al. [47] reports that oftentimes microservices are dependent on third-party libraries and services. If the third-party libraries and services are vulnerable, it can have a negative effect on the microservices if they are not patched. It is important to understand the dependencies between microservices and third-party libraries used in the development of microservices.

3.1.6.2. Security Practices

Newman [8] describes the importance of knowing who owns the infrastructure and software that microservices run on and assigning the right personnel to maintain and handle the patching. Venčkauskas et al. [47] makes a similar suggestion as Newman but on the third-party libraries and services that microservices use in their development.

3.1.7. Deployment

3.1.7.1. Challenges

Siriwardena and Dias [26] describes as the deployment of microservices increases in scale, it makes it extremely challenging to manage and maintain the security. Each communication channel between microservices requires protection. Each microservice must manage the authentication, authorization, revocation, and rotation of the security mechanism when interacting with another microservice. Two major financial institutions in the United States and United Kingdom are mentioned to illustrate the deployment scale. In July 2019, Capital One deployed thousands of microservices on several thousands of containers with thousands of Amazon Elastic Compute Cloud instances. In November 2019, Monzo had more than 1500 services running on its microservices deployment. Without a way to automate security, it makes it extremely different to manage microservices in a large-scale deployment.

Siriwardena and Dias [26] also describes the challenge of maintaining service credentials and access control policies in containers. If a microservice is deployed in a container, the container is considered as an immutable container. The container is booted up from a base configuration. Any changes to the files in the file system and the runtime state are not maintained by the container. It means that any changes to the clients and access-control policies that were previously updated in an instance of a microservice will not be sharable to another instance of the same microservice. It creates an issue with how service credentials and access control policies are maintained across different instances of the same microservice.

Torkura et al. [2] states that it is typically for different development teams to be in charge of building microservices that serve different business needs. Development teams will use the most appropriate technologies in the development of the microservices based on the team's development experience and the business requirements. While this development pattern aids productivity, it makes managing security more challenging since different technologies have different security concerns and vulnerabilities. The technology agnostic nature of microservices also makes vulnerability detection more difficult [67]. Joseph and Chandrasekaran [64] states that the number of security capabilities are higher due to the polyglot stack functionality of microservices.

The dynamic deployment of microservices results in constant changes in resource parameters, e.g., IP addresses, port numbers, and service endpoints. The constant changes in resource parameters poses a challenge in security assessments which are traditionally configured for static network resources [2].

3.1.7.2. Security Practices

Regarding the issue of immutability of containers and how service credentials and access control policies are maintained, Siriwardena and Dias [26] suggests the use of a push or a pull model. The service credentials and access control policies are maintained at a policy administration endpoint. With a push model, the policy administration endpoint pushes the updates to the microservice at bootup. With a pull model, the microservice periodically pulls updates from the policy administration endpoint.

3.1.8. Trust

3.1.8.1. Challenges

Dragoni et al. [36] describes that microservices are often designed to trust each other in a microservice architecture. Microservices architecture is vulnerable to both threats from other hosts and threats from components within the boundary of the system [64]. When a malicious adversary attacks and gains control of an individual microservice, it can affect other microservices in the microservice architecture. The malicious adversary can manipulate microservices to do what he/she wants them to do, escalate privileges on the hosting infrastructure of the microservices, listen on any inter-service communication, alter data in transit, lead to full disclosure of other microservices, and potentially bring down the entire system [18, 4, 32, 36, 2].

Yuqiong et al. [32] describes a real-world example on how trust relationships between microservices can have a negative impact on the security of a microservice architecture. A subdomain of Netflix was compromised, it led to an adversary having the ability to serve any content on netflix.com and ability to tamper with authenticated Netflix subscribers and their data since Netflix allowed all users' cookies to be accessed from any subdomain.

3.1.8.2. Security Practices

Mateus-Coelho et al. [4] discusses the importance of providing layers of security throughout an architecture and suggests the use of firewalls, intrusion detection systems, and intrusion prevention systems as defense mechanisms on top of other security mechanisms.

Newman [8] suggests the use of the principle of zero trust where the environment is hostile and bad actors could be present to launch an attack and threat modeling to drive the security design in a microservice infrastructure. He provides an example of a secure design used in a healthcare system where sensitive data is kept. The data is classified based on their sensitivity level. Microservices are classified based on the most sensitive data they use. Each microservice runs in the zone matching the most sensitive data it uses. Microservices in the same zone can communicate without each other. Microservices in a more secure zone can use microservices in a lower secure zone.

Venčkauskas et al. [47] suggests dividing microservices according to the degree of access they need, with stricter security measures for the critical microservices. That way, the damage can be isolated instead of propagated to the rest of the system.

3.1.9. Larger Surface Area

3.1.9.1. Challenges

Microservices architecture is a style where a software application is built as a composition of microservices, and microservices communicate with each other via APIs over the network. Communications between microservices over the network cause exposure to more potential attacks than a monolithic application due to the increased number in entry points, and hence increases the attack surface area [36, 3, 2, 26, 19, 4, 47]. With the attack surface area being larger, it makes it harder to manage security.

3.1.9.2. Security Practices

API Gateway

Torkura et al. [2] describes the use of continuous security assessments with security gateway, dynamic document stores, and security health endpoints to reduce attack surfaces. Security gateway serves as a security enforcement point that enforces security policies on microservices and infrastructure components. Each microservice is designed to provide an openAPI document. Dynamic document stores openAPI documents of microservices which allows security scanners to extract information for security testing. Security health endpoints provide security status and assessment results of microservices.

Zero Trust Model

Newman [8] suggests the use of the principle of zero trust where the environment is hostile and bad actors could be present to launch an attack and threat modeling to drive the security design in a microservice infrastructure. He provides an example of a secure design used in a healthcare system where sensitive data is kept. The data is classified based on their sensitivity level. Microservices are classified based on the most sensitive data they use. Each microservice runs in the zone matching the most sensitive data it uses. Microservices in the same zone can communicate without each other. Microservices in a more secure zone can use microservices in a lower secure zone.

3.2. Development of the Framework for Security Modeling and Specification in Microservice Architectures

This section describes the development of the approach, called Framework for Security Modeling and Specification in Microservice Architectures, to enhance the security modeling and specifications in microservice architectures (RQ3). The basis for the approach is the review of the literature, documentation, and industry practices described in Section 3.1. The approach leverages the security challenges in microservices architecture and integrates the corresponding security practices to create a set of security properties for microservice architectures in AADL and a set of decision trees to guide software architects on how to use the security properties when designing a secure microservice architecture and what additional structural elements (components and connectors) are required to support a secure architecture design. With the security properties annotated on components and connectors, it allows software architects to run analysis and simulations at the architecture level to ensure that security architecture characteristics are satisfied.

The next subsections present the security properties grouped as follows: general security properties, network perimeter related properties, edge level related properties, communication related properties, data related properties, log related properties, deployment related properties, trust related properties, authentication related properties, and authorization related properties. The decision trees associated with each property can be found in Appendix A.

3.2.1. General Security Properties

3.2.1.1. SecurityLevelProvided

- **Property Name:** SecurityLevelProvided
- **Description:**
    - This property specifies the level of security the component or connector offers.
    - This property is an enumeration with the values of networkPerimeterLevel, edgeLevel, serviceLevel, communicationLevel, infrastructureLevel. The semantics of the values are:
        - networkPerimeterLevel refers to the component offering network perimeter security and enforcing network perimeter related security rules.
        - edgeLevel refers to the component offering edge security and enforcing edge related security rules.
        - serviceLevel refers to the component that is representing an individual core microservice offering security at the microservice level. The security rules

        are enforced by the individual core microservices. Core microservice refers to microservice that serves a business purpose in the given problem domain.
        - infrastructureLevel refers to the component that is not representing an individual core microservice offering security at the microservice level and enforcing security rules that will impact one or more individual core microservices in the microservice architecture. Core microservice refers to microservice that serves a business purpose in the given problem domain.
        - communicationLevel refers to the connector offering communication security and enforcing communication related security rules.
    - This property can be specified for abstract, system, process, thread, and connection.
    - This property is derived from the concept of defense in depth concept where security should be applied and layered throughout the microservice architecture. The concept of defense in depth is described in the following literatures:
        - [8]
        - [26].
- **Declaration:**

  SecurityLevelProvided: enumeration (networkPerimeterLevel, edgeLevel, serviceLevel, communicationLevel, infrastructureLevel) applies to (abstract, system, process, thread, connection);
- **Property Association Example:**
    - SecurityLevelProvided => networkPerimeterLevel;

3.2.2. Network Perimeter Related Properties

3.2.2.1. NetworkPerimeterLevel

- **Property Name:** NetworkPerimeterLevel
- **Description:**

    - This property specifies the type of network perimeter security mechanism that the component provides.
    - This property is an enumeration with the values of intrusionDetectionPrevention and firewallAccess. The semantics of the values are:
        - intrusionDetectionPrevention refers to the ability to detect irregular and unusual activities.
        - firewallAccess refers to the ability to accept and/or deny requests based on IP addresses.
    - This property can be specified for abstract, system, process, and thread.
    - This property is derived from the following literatures:
        - [8]
        - [26].
- **Declaration:**

NetworkPerimeterLevel: enumeration (intrusionDetectionPrevention, firewallAccess) applies to (abstract, system, process, thread);

- **Property Association Example:**
    - NetworkPerimeterLevel => intrusionDetectionPrevention;
    - NetworkPerimeterLevel => firewallAccess;

3.2.3. Edge Level Related Properties

3.2.3.1. EdgeLevel

- **Property Name:** EdgeLevel
- **Description:**
    - This property specifies the type of edge security mechanism that the component provides.

- This property is an enumeration with the value of APIGateway. The semantics of the value is:
  - APIGateway [57, 30, 26] is a reverse proxy that is typically deployed at the edge of a system, data center, or as part of each product, line of business, or department, or between a public network and demilitarized zone of a private network. It serves as an entry point for a defined group of APIs. It decouples external APIs from internal microservice APIs and prevents microservices from being contacted directly.
  - API gateway protects APIs from overuse and abuse, such as with throttle limits, it reduces the chance of DoS/DDoS attacks [57, 30]
- This property can be specified for abstract, system, process, and thread.
- This property is derived from the following literatures:
  - [57]
  - [30]
  - [37]
  - [26].

- **Declaration:**

EdgeLevel: enumeration (APIGateway) applies to (abstract, system, process, thread)

- **Property Association Example:**
  - EdgeLevel => APIGateway;

3.2.3.2. APIRequestPerSecond_type

- **Property Name:** APIRequestPerSecond_type
- **Description:**
  - This is a property type that defines the maximum rate of requests a single API can receive per second.

- This is a property of type "type record" with two fields:
- APIName: This field is of type aadlstring. It defines the name of the API that fulfills a request.
- requestPerSecond: This field is of type aadlinteger and must be a positive numeric value. It defines the rate of requests per second for the said API.
- This property is created to support the structure of the following properties:
    - MaxRequestMicroservice
    - APIRequestPerSecondMicroserviceName_type.

- **Declaration:**

```
APIRequestPerSecond_type: type record (
    APIName: aadlstring;
    requestPerSecond: aadlinteger units (perSecond);
);
```

- **Property Association Example:** Not available because it is a property type.

3.2.3.3. APIRequestPerSecondMicroserviceName_type

- **Property Name:** APIRequestPerSecondMicroserviceName_type
- **Description:**
    - This is a property type that defines a list of the maximum rate of requests a single API can receive per second for all APIs that a microservice has.
    - This is a property of type "type record" with two fields:
        - microserviceName: This field is of type aadlstring. It defines the name of the microservice that owns the APIs.
        - APIRequest: This field is a list of APIRequestPerSecond_type values.
    - This property is created to support the structure of the following properties:
        - MaxRateLimitEdge

        - APIRequestPerSecondMicroserviceNameApp_type
        - APIRequestPerSecondMicroserviceNameUser_type
        - APIRequestPerSecondMicroserviceNameOperation_type.
- **Declaration:**

```
APIRequestPerSecondMicroserviceName_type: type record(
        microserviceName: aadlstring;
        APIRequest: list of
        Microservice_Architecture_Security_Properties::APIRequestPerSecond_type;
);
```

- **Property Association Example:** Not available because it is a property type.

3.2.3.4. MaxRateLimitEdge

- **Property Name:** MaxRateLimitEdge
- **Description:**
    - This property specifies the rate limiting. Rate limiting refers to if the rate of incoming request exceeds a predefined number of requests per second, the edge level security microservice will reject all the incoming requests. It is to handle unprecedented surges and to prevent total loss of availability at the edge. The trade off is to serve as many requests as possible instead of encountering service unavailability for everyone.
    - This property is a list of APIRequestPerSecondMicroserviceName_type values.
    - This property must include all the APIs that are exposed at the edge.
    - This property can be specified for abstract, system, process, and thread.
    - This property is related to MaxRequestMicroservice. Please refer to MaxRequestMicroservice property for more information.
    - This property is derived from the following literatures:

        - [66]
        - [26]
        - [58].

- **Declaration:**

MaxRateLimitEdge: list of Microservice_Architecture_Security Properties:APIRequestPerSecondMicroserviceName_type applies to (abstract, system, process, thread);

- **Property Association Example:**
    - MaxRateLimitEdge =>

```
([microserviceName => "n";
 APIRequest=>
        ([APIName => "m"; requestPerSecond=> p perSecond;],
        ...,
        [APIName => "q"; requestPerSecond=> r perSecond;])],
         …,
 [microserviceName => "a";
 APIRequest=>
        ([APIName => "b"; requestPerSecond=> c perSecond;],
        ...,
        [APIName => "d"; requestPerSecond=> e perSecond;])]);
```

-- where n/a is a string that represents the name of the microservice

-- where m/b/q/d is a string that represents the name of the API

-- where p/r/c/e is an integer that represents the maximum number of requests an API can process per second

    - MAXRateLimitEdge =>

```
([microserviceName => "Order";
APIRequest=>
```

```
        ([APIName => "View_Order"; requestPerSecond=> 45 perSecond;],
        [APIName => "Place_Order"; requestPerSecond=> 45 perSecond;]);],
    [microserviceName => "Catalog";
    APIRequest=>
        ([APIName => "View_Catalog"; requestPerSecond=> 45 perSecond;]);]);
```

3.2.3.5. MaxRequestMicroservice

- **Property Name:** MaxRequestMicroservice
- **Description:**
    - This property defines the maximum rate of requests per second that a microservice can handle for all the APIs the microservice has.
    - This property is a list of APIRequestPerSecond_type values.
    - This property can be specified for abstract, system, process, and thread.
    - This property is derived from the following literatures:
        - [26]
        - [58].
- **Declaration:**

```
MaxRequestMicroservice: list of
Microservice_Architecture_Security_Properties::APIRequestPerSecond_type
applies to (abstract, system, process, thread);
```

- **Property Association Example:**
    - maxRequestMicroservice =>

```
        ([APIName => "p", requestPerSecond= q perSecond],
                ...,
        [APIName => "q", requestPerSecond=b perSecond]);
    -- where p/a is a name of the API
```

-- where q/b is an integer representing the maximum number of requests that said API can handle per second.

- MaxRequestMicroservice =>

  ([APIName => "View_Order"; requestPerSecond=> 50 perSecond;],

  [APIName => "Place_Order"; requestPerSecond=> 50 perSecond;]);

3.2.3.6. APIRequestPerSecondMicroserviceNameApp_type

- **Property Name: APIRequestPerSecondMicroserviceNameApp_type**
- **Description:**
  - This is a property type that defines a list of the maximum rate of requests a single API can receive per second for the APIs that a microservice has exposed at the edge for each application type.
  - This is a property of type "type record" with two fields:
  - applicationType: This field is of type aadlstring. It defines the origin of where the external requests come from. The typical values are web application, mobile application, and tablet application.
  - APIRequestMicroservice: This field is a list of APIRequestPerSecondMicroserviceName_type.
  - This property is created to support the structure of the following property:
    - RequestPerApplicationType.
- **Declaration:**

APIRequestPerSecondMicroserviceNameApp_type: type record (

applicationType: aadlstring;

APIRequestMicroservice: list of

Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroserv

iceName_type;

);

- **Property Association Example:** Not available because it is a property type.

3.2.3.7. APIRequestPerSecondMicroserviceNameUser_type

- **Property Name:** APIRequestPerSecondMicroserviceNameUser_type
- **Description:**
    - This is a property type that defines a list of the maximum rate of requests a single API can receive per second for APIs that a microservice has exposed at the edge for each user.
    - This is a property of type "type record" with two fields:
        - userIdentifier: This field is of type aadlstring. It defines who is sending the external requests.
        - APIRequestMicroservice: This field is a list of APIRequestPerSecondMicroserviceName_type.
    - This property is created to support the structure of the following property:
    - RequestPerUser.
- **Declaration:**

```
APIRequestPerSecondMicroserviceNameUser_type: type record(
        userIdentifier: aadlstring;
        APIRequestUser: list of
        Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroserv
        iceName_type;
);
```

- **Property Association Example:** Not available because it is a property type.

3.2.3.8. APIRequestPerSecondMicroserviceNameOperation_type

- **Property Name:** APIRequestPerSecondMicroserviceNameOperation_type
- **Description:**
    - This is a property type that defines a list of maximum rate of requests an operation of an abstracted API can receive per second for APIs that a microservice has exposed at the edge. An API can be an abstraction of one or more operations. This property allows the regulation of the maximum rate of request to be done at the operation level.
    - This is a property of type "type record" with two fields:
        - operationName: This field is of type aadlstring. It defines the name of the operation that needs regulation on the maximum rate of requests it can process.
        - APIRequestMicroservice: This field is a list of APIRequestPerSecondMicroserviceName_type.
    - This property is created to support the structure of the following property:
    - RequestPerOperation.
- **Declaration:**

```
APIRequestPerSecondMicroserviceNameOperation_type: type record (
        operationName: aadlstring;
        APIRequestOperation: list of
        Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroservice
        Name_type;
    );
```

- **Property Association Example:** Not available because it is a property type.

3.2.3.9. RequestPerApplicationType

- **Property Name:** RequestPerApplicationType
- **Description:**
    - This property specifies the maximum rate of requests per second that the edge security component should handle per application type. If the rate of requests exceeds a predefined number of requests per second, the edge level security microservice will wait for a duration of time before processing more requests.
    - This property lowers the risk of malicious attacks like denial of service/distributed denial of service attacks where targeted resources are being overloaded with traffic and unable to perform their responsibilities.
    - This is a property of type "record" with 3 fields:
        - description: This field is of type aadlstring. It provides a description of the usage of the property.
        - waitTime: This field is of type Time and the values must be positive. The standard Time units are ps (picoseconds), ns (nanoseconds), us (microseconds), ms (milliseconds), sec (seconds), min (minutes), and hr (hours).
        - APIRequestPerSecondMicroserviceNameApp_type: This field is a list of APIRequestPerSecondMicroserviceNameApp_type.
    - This property can be specified for abstract, system, process, and thread.
    - This property is derived from the following literature:
        - [26]
        - [58].
- **Declaration:**

RequestPerApplicationType: record (

    description: aadlstring;

```
    waitTime: Time;
    microservice_API_requestPerSecond_App: list of
    Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroserviceNa
    meApp_type;
) applies to (abstract, system, process, thread);
```

- **Property Association Example:**
  - RequestPerApplicationType => [

```
description => "n";
waitTime => z sec;
microservice_API_requestPerSecond_App =>
        ([applicationType => "m";
         APIRequestMicroservice =>
                ([microserviceName => "a";
                  APIRequest =>
                        ([APIName => "b";
                        requestPerSecond => c perSecond;],
                         ...
                          [APIName => "d";
                        requestPerSecond => d perSecond;]);],
                [microserviceName => "f";
                APIRequest =>
                        ([APIName => "g";
                        requestPerSecond => h perSecond;],
                        ...
                        [APIName => "i";
                        requestPerSecond => j perSecond;]);]);],
[applicationType => "k";
APIRequestMicroservice =>
               ([microserviceName => "l";
                APIRequest =>
                       ([APIName => "o";
                         requestPerSecond => p perSecond;],
                          ...
                       [APIName => "q";
                       requestPerSecond => r perSecond;]);]);]);];
```

-- where n is a series of strings that describes limitation of requests at the application level

-- where m/k is a string that describes the application type

-- where a/f/l is a string that represents the name of the microservice

-- where b/d/g/i/o/q is a string that represents the name of the API

-- where c/e/h/j/p/r/z is an integer in seconds

- o RequestPerApplicationType =>

```
[description => "Regulation rate of request by mobile application'";
waitTime => 1000 sec;
microservice_API_requestPerSecond_App =>
            ([applicationType => "Mobile application";
            APIRequestMicroservice =>
                        ([microserviceName => "Order";
                          APIRequest =>
                                    ([APIName => "View_Order";
                                    requestPerSecond => 50 perSecond;],
                                    [APIName => "Place_Order";
                                    requestPerSecond => 50 perSecond;]);],
                         [microserviceName => "Catalog";
APIRequest =>
        ([APIName => "View_Catalog";
        requestPerSecond => 50 perSecond;]);]);]);];
```

3.2.3.10. RequestPerUser

- **Property Name:** RequestPerUser
- **Description:**
    - This property specifies the maximum rate of requests per second that the edge security component should handle per user. If the rate of requests exceeds a predefined number of requests per second, the edge level security microservice will wait for a duration of time before processing more requests.
    - This property lowers the risk of malicious attacks like denial of service/distributed denial of service attacks where targeted resources are being overloaded with traffic and unable to perform their responsibilities.
    - This is a property of type "record" with 3 fields:
        - description: This field is of type aadlstring. It provides a description of the usage of the property.
        - waitTime: This field is of type Time and the values must be positive. The standard Time units are ps (picoseconds), ns (nanoseconds), us (microseconds), ms (milliseconds), sec (seconds), min (minutes), and hr (hours).
        - APIRequestPerSecondMicroserviceNameApp_type: This field is a list of APIRequestPerSecondMicroserviceNameUser_type.
    - This property can be specified for abstract, system, process, and thread.
    - This property is derived from the following literature:
        - [26]

        - [58].

- **Declaration:**

```
RequestPerUser: record (
    description: aadlstring;
    waitTime: Time;
    microservice_API_requestPerSecond_User: list of
    Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroserviceNa
    meUser_type;
) applies to (abstract, system, process, thread);
```

- **Property Association Example:**
    - RequestPerUser => [

```
description => "n";
waitTime => z sec;
microservice_API_requestPerSecond_User =>
        ([userIdentifier => "m";
         APIRequestUser =>
                ([microserviceName => "a";
                APIRequest =>
                        ([APIName => "b";
                        requestPerSecond => c perSecond;],
                          ...
                        [APIName => "d";
                        requestPerSecond => d perSecond;]);],
                 [microserviceName => "f";
                APIRequest =>
                        ([APIName => "g";
                        requestPerSecond => h perSecond;],                    ...
```

```
                    [APIName => "i";
                    requestPerSecond => j perSecond;]);]);],
         [userIdentifier => "k";
         APIRequestUser =>
              ([microserviceName => "l";
              APIRequest =>
                   ([APIName => "o";
                   requestPerSecond => p perSecond;],
                    ...
                   [APIName => "q";
                   requestPerSecond => r perSecond;]);]);]);];
```

-- where n is a series of strings that describes limitation of requests at the user level

-- where m/k is a string that describes the user identity

-- where a/f/l is a string that represents the name of the microservice

-- where b/d/g/i/o/q is a string that represents the name of the API

-- where c/e/h/j/p/r/z is an integer in seconds

o RequestPerUser =>

```
     [description => "Regulation rate of request by user identifier'";
     waitTime => 1000 sec;
     microservice_API_requestPerSecond_User =>
          ([userIdentifier => "abewerwerewrwrwerwerrew";
           APIRequestUser =>
               ([microserviceName => "Order";
                APIRequest =>
                    ([APIName => "View_Order";
                    requestPerSecond => 50 perSecond;],
                    [APIName => "Place_Order";
```

```
        requestPerSecond => 50 perSecond;]);],
    [microserviceName => "Catalog";
    APIRequest =>
        ([APIName => "View_Catalog";
        requestPerSecond => 50 perSecond;]);]);]);];
```

3.2.3.11. RequestPerOperation

- **Property Name:** RequestPerOperation
- **Description:**
  - This property specifies the maximum rate of requests per second that the edge security component should handle per operation. If the rate of requests exceeds a predefined number of requests per second, the edge level security microservice will wait for a duration of time before processing more requests.
  - This property lowers the risk of malicious attacks like denial of service/distributed denial of service attacks where targeted resources are being overloaded with traffic and unable to perform their responsibilities.
  - This is a property of type "record" with 3 fields:
    - description: This field is of type aadlstring. It provides a description of the usage of the property.
    - waitTime: This field is of type Time and the values must be positive. The standard Time units are ps (picoseconds), ns (nanoseconds), us (microseconds), ms (milliseconds), sec (seconds), min (minutes), and hr (hours).
    - APIRequestPerSecondMicroserviceNameApp_type: This field is a list of APIRequestPerSecondMicroserviceNameOperation_type.
  - This property can be specified for abstract, system, process, and thread.

- This property is derived from the following literatures:
  - [26]
  - [58].
- **Declaration:**

```
RequestPerOperation: record (
    description: aadlstring;
    waitTime: Time;
    microservice_API_requestPerSecond_Op: list of
    Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroserviceNa
    meOperation_type;
) applies to (abstract, system, process, thread);
```

- **Property Association Example:**
  - RequestPerOperation => [

```
        description => "n";
        waitTime => z sec;
        microservice_API_requestPerSecond_Op =>
                ([operationName => "m";
                APIRequestOperation =>
                        ([microserviceName => "a";
                        APIRequest =>
                                ([APIName => "b";
                                  requestPerSecond => c perSecond;],
                                  ...
                                 [APIName => "d";
                                requestPerSecond => d perSecond;]);],
                        [microserviceName => "f";
                        APIRequest =>
```

```
                    ([APIName => "g";
                    requestPerSecond => h perSecond;],                 ...
                    [APIName => "i";
                    requestPerSecond => j perSecond;]);]);],
         [userIdentifier => "k";
        APIRequestOperation =>
                ([microserviceName => "l";
                APIRequest =>
                    ([APIName => "o";
                    requestPerSecond => p perSecond;],
                     ...
                    [APIName => "q";
                    requestPerSecond => r perSecond;]);]);]);];
-- where n is a series of strings that describes limitation of requests at the user level
-- where m/k is a string that describes the user identity
-- where a/f/l is a string that represents the name of the microservice
-- where b/d/g/i/o/q is a string that represents the name of the API
-- where c/e/h/j/p/r/z is an integer in seconds
  o  RequestPerUser =>
        [description => "Regulation rate of request by user identifier'";
        waitTime => 1000 sec;
        microservice_API_requestPerSecond_Op =>
               ([operationName => "";
                APIRequestMicroservice =>
                    ([microserviceName => "Order";
                      APIRequest =>
                          ([APIName => "View_Order";
                    requestPerSecond => 50 perSecond;],
                     [APIName => "Place_Order";
                    requestPerSecond => 50 perSecond;]);],
          [microserviceName => "Catalog";
                APIRequest =>
                    ([APIName => "View_Catalog";
                    requestPerSecond => 50 perSecond;]);]);]);];
```

3.2.3.12. MessagePayloadSizeLimit

- **Property Name:** MessagePayloadSizeLimit
- **Description:**
    - This property specifies the maximum message payload size an external application can send.
    - This property lowers the risk of malicious attacks like denial of service/distributed denial of service attacks where targeted resources are being overloaded with traffic and unable to perform their responsibilities.
    - This property is of type aadlinteger with mb as the unit.
    - This property can be specified for data and port.
    - This property is derived from the following literature:
        - [58].
- **Declaration:**

MessagePayloadSizeLimit: aadlinteger units (mb) applies to (data, port);

- **Property Association Example:**
    - MessagePayloadSizeLimit => n;

      -- where n is an integer with units in mb.
    - MessagePayloadSizeLimit => 10 mb;

3.2.4. Communication Related Properties

3.2.4.1. SecureCommunication

- **Property Name:** SecureCommunication
- **Description:**
  - This property specifies the communication protocol used between microservices.
  - This property is an enumeration with the values of TLS and MTLS. The semantics of the values are:
    - TLS: It stands for transport layer security. It protects the communication between two microservices because the upstream (client) microservice knows the identity of the downstream (server) microservice it interacts with. Messages are protected for integrity and confidentiality while in transit.
    - MTLS: It stands for mutual transport layer security. It protects the communication between two microservices because the upstream (client) microservice knows the identity of the downstream (server) microservice it is interacting with and the downstream (server) microservice knows the identity of the upstream (client) microservice it interacts with. Messages are protected for integrity and confidentiality while in transit.
  - This property can be specified only for connectors.
  - This property and the property values are derived from the literatures listed in Section 3.1.4. Communications.
- **Declaration:**

SecureCommunication: enumeration (TLS, MTLS) applies to (connection);

- **Property Association Example:**
  - SecureCommunication => TLS;
  - SecureCommunication => MTLS;

3.2.5. Data Related Properties

3.2.5.1. dbAccessModel

- **Property Name:** dbAccessMode
- **Description:**
  - This property specifics the access mode to the database of a microservice.
  - This property is an enumeration with the values of read_only, read_write, write_only. The semantics of the values are:
    - read_only refers to read access to the database that is associated with the microservice.
    - read_write refers to read write access to the database that is associated with the microservice.
    - write_only refers to write access to the database that is associated with the microservice.
  - This property can be specified for data.
  - This property and the property values are derived from the literatures listed in Section 3.1.5. Data.
- **Declaration:**

dbAccessMode: enumeration (read_only, read_write, write_only) applies to (data);

- **Property Association Example:**
  - dBAccessMode => n; -- where n is either "read_only, "read_write", or "write_only"
  - dBAccessMode => read_only;

3.2.5.2. DataAtRest

- **Property Name:** DataAtRest
- **Description:**

- This is a property type that defines the basic building block of data encryption for data at rest.
- This is a property of type "type record" with 4 fields:
    - description: This field is of type aadlstring. It describes the need to encrypt or not encrypt a data block.
    - dataBlock: This field is of type aadlstring. It describes the size and scope of the data block that might or might not require encryption.
    - dataCriticality: This field is of type aadlinteger and must be a positive numeric value. The higher the value, the more critical the data in the data block is.
    - atRestEncryption: This field is an enumeration with the values required and not required. The semantics of the values are:
        - required refers to encryption is required on the specified data block.
        - not_required refers to encryption is not required on the specified data block.
- This property is created to support the structure of the following property:
    - DataAtRestEncryption.

- **Declaration:**

```
DataAtRest: type record(
        description: aadlstring;
        dataBlock: aadlstring;
        dataCriticality: aadlinteger;
        atRestEncryption: enumeration (required, not_required);
);
```

- **Property Association Example:** Not available because it is a property type.

3.2.5.3. DataAtRestEncryption

- **Property Name:** DataAtRestEncryption
- **Description:**
  - This property specifies a list of data blocks and their requirements on encryption.
  - This property is a list of DataAtRest values.
  - This property can be specified for data.
  - This property and the property values are derived from the literatures listed in Section 3.1.5. Data.
- **Declaration:**

DataAtRestEncryption: list of Microservice_Architecture_Security_Properties::DataAtRest applies to (data);

- **Property Association Example:**
  - DataAtRestEncryption => (

```
[description => "m";
dataBlock => "p";
dataCriticality => q;
atRestEncryption => r;],
[description => "s";
dataBlock => "t";
dataCriticality => u;
atRestEncryption => v;],);
```

-- where m/s is a string that describes the need to encrypt or not encrypt a data block.

-- where p/t is a string that describes the size and scope of the data block that might or might not require encryption.

-- where q/u is an integer that describes the criticality of the data in the data block. The higher the number, the more critical it is.

-- where r/v is either "required" or "not_required".

3.2.5.4. MicroserviceDataSensitivity

- **Property Name:** MicroserviceDataSensitivity
- **Description:**
  - This property specifies the relationship between microservices from a data perspective to facilitate the proper flow of data and communication between microservices.
  - Zone is a way to classify microservices within the same trust domain so that communication patterns among microservices can be established. Within the same trust domain,
    - Microservices within the same zone* can communicate with each other.
    - Microservice in a higher zone can communicate with microservices that are in a lower zone.
    - Microservice in a lower zone cannot communicate with microservices that are in a higher zone.
  - This is a property of type "record" with 3 fields:
    - description: This field is of type aadlstring. It provides a description of the usage of the property.
    - dataSenstivityLevel: This field is an enumeration with the values of publicData, privateData, secretData, and no_Data. The semantics of the values are:
    - publicData: It refers to data that can be used and accessed by anyone without restriction.
    - privateData: It refers to data that can only be used and accessed by authorized users.

        - secretData: It refers to data that can only be used and accessed by a specific list of authorized users.
        - no_Data: It refers to the microservice that does not own or store data.
    - This property can be specified for abstract, system, process, and thread.
    - This property and the property values are derived from the literatures listed in Section 3.1.5. Data.
- **Declaration:**

```
MicroserviceDataSensitivity: record (
        description: aadlstring;
        dataSensitivityLevel: enumeration (publicData, privateData, secretData, no_Data);
        microserviceZone: aadlinteger;
) applies to (abstract, system, process, thread);
```

- **Property Association Example:**
    - MicroserviceDataSensitivity => [

```
                description => "n";
                dataSensitivityLevel => m;
                microserviceZone => p;
        ]
    -- where n a unique integer corresponding to  the zone
    -- where n is a series of strings that describes the data sensitivity level
    -- where m is either "publicData", "privateData", or  "secretData, no_Data"
```

3.2.6. Log Related Properties

3.2.6.1. Log

- **Property Name:** Log
- **Description:**

- This property specifies the requirements on logging.
- This is a property of type "record" with 4 fields:
    - description: This field is of type aadlstring. It describes the logging needs of the microservice.
    - logFormat: This field is of type aadlstring. It describes the format requirement of the log in facilitating the aggregation of logs and queries against aggregated logs.
    - logAttribute: This field is of type aadlstring. It describes the information that a microservice should collect, such as correlation ID, date, time in a specific format and time zone.
    - logLevel: This field is an enumeration with the values of trace, debug, information, warning, error, and critical. The semantics of the values are:
        - trace: It captures all the details of the behavior of the microservice.
        - debug: It captures diagnostic information.
        - information: It captures normal behavior of the microservice.
        - warning: It captures unexpected behavior of the microservice.
        - error: It captures error messages.
        - critical: It captures fatal error messages that cause the crushing of the microservice.
- This property can be specified for abstract, system, process, and thread.
- This property and the property values are derived from the literatures listed in Section 3.1.5. Data.

- **Declaration:**

```
Log: record (
    description: aadlstring;
    logFormat: aadlstring;
    logAttribute: list of aadlstring;
```

logLevel: enumeration (trace, debug, information, warning, error, critical);

) applies to (abstract, system, process, thread);

- **Property Association Example:**
  - Log => [

    description => "n";

    logFormat => "m";

    logAttribute => ("p", ..., "a");

    logLevel => q;)

    ];

    -- where n is a series of strings that describes the details of the log aggregator

    -- where m is a series of strings that describes the structure of the log

    -- where p/a is a series of strings that describes the attributes to log

    -- where q is either "trace", "debug", "information", "warning", "error", or "critical"

3.2.7. Deployment and Patching Related Properties

3.2.7.1. patch_type

- **Property Name:** patch_type
- **Description:**
  - This is a property type that defines the patching frequency of a specific software.
  - This is a property of type "type record" with two fields:
    - softwareName: This field is of type aadlstring. It defines the name of the software that requires patching.
    - frequency: This field is of type aadlstring. It defines how often patching is request for the said software.
  - This property is created to support the structure of the following property:
    - DeploymentType.

- **Declaration:**

```
patch_type: type record (
        softwareName: aadlstring;
        frequency: aadlstring;
);
```

- **Property Association Example:** Not available because it is a property type.

3.2.7.2. DeploymentType

- **Property Name:** DeploymentType
- **Description:**
    - This property specifies the deployment and patching configuration of a microservice.
    - This property is of type "record" with 4 fields:
        - description: This field is of type aadlstring. It describes the deployment and patching configuration plan.
        - deploymentMechanism: This field is an enumeration with the values physical_Machine, virtual_machine, container, platform_as_a_service, and function_as_a_service. The semantics of the values are:
        - physical_Machine: Physical machine deployment option refers to deploying a single microservice on a physical machine without any layers of virtualization or containerization between the microservice and underlying hardware. Deploying multiple microservices on the same physical machine violates the isolated execution environment principle.
        - virtual_machine: Virtual machine deployment option refers to deploying a single microservice on a virtual machine. The virtual machine contains a full operating system, kernel, and resources that a microservice can use.

        - container: Container deployment option refers to deploying a single microservice on a container. A container contains resources that a microservice can use.
        - platform_as_a_service: Platform as a service deployment (PaaS) option refers to deploying a single microservice on a cloud infrastructure. Platform as service includes infrastructure as a service (IaaS) (computing, networking, and storage resources) and resources such as development tools, database management systems, middleware,and notification systems [56]. The exact amount of resource provided depends on the provider of the platform as a service. Examples of IaaS are AWS, Microsoft Azure, and Google Compute Engine. Examples of PaaS are AWS Elastic Beanstalk, Google App Engine, and Heroku.
        - function_as_a_service: Function as a service deployment (FaaS) option refers to deploying a single microservice as a function on the cloud infrastructure. Example of FaaS is AWS's Lambda product.
    - patchList: This field is a list of patch_type.
  - This property can be specified for abstract, system, process, and thread.
  - This property and the property values are derived from the literatures listed in Sections 3.1.6. Patching and 3.1.7. Deployment.
- **Declaration:**

```
DeploymentType: record (
        description: aadlstring;
        deploymentMechanism: enumeration (physical_Machine, virtual_machine, container,
        platform_as_a_service, function_as_a_service);
        patchList: list of Microservice_Architecture_Security_Properties::patch_type;
) applies to (abstract, system, process, thread);
```

- **Property Association Example:**

```
o   DeploymentType => [
            description => "n";
            deploymentMechanism => m;
            patchList => (
                    [softwareName => p; frequency => q;],
                    ...,
                    [softwareName => a; frequency =>b;]
            )
    ];
```

-- where n is a series of strings that describes the deployment mechanism of the microservice

-- where m is either "physical _machine", "virtual_machine", "container", "platform_as_a_service", or "function_as_a_service"

-- where p/a is a name of the software that needs to be patched and q/b is the frequency in which p/a requires patching.

```
o   Deployment_Type => [
            description => "Deployment and patching configuration of Catalog";
            deploymentMechanism => virtual_machine;
            patchList => ([softwareName => "PackageX"; frequency => "2 months";]);
    ];
```

3.2.8. Trust Related Properties

3.2.8.1. TrustDomain

- **Property Name:** TrustDomain
- **Description:**

- This property specifies the trust domain a microservice belongs to. Each microservice can only belong to one trust domain.
- This property is of type aadlinteger and must be a positive number.
- This property can be specified for abstract, system, process, and thread.
- This property is derived from the literatures listed in Section 3.1.8. Trust.

- **Declaration:**

TrustDomain: aadlinteger applies to (abstract, system, process, thread);

- **Property Association Example:**
  - TrustDomain => n; -- where n a unique integer corresponding to the trust domain
  - TrustDomain => 2;

3.2.9. Authentication Related Properties

3.2.9.1. AuthenticationMicroserviceArchitecture

- **Property Name:** AuthenticationMicroserviceArchitecture
- **Description:**
  - This property specifies the authentication mechanism used in the microservice architecture.
  - This property is an enumeration with the values APIGateway, token_based_authentication, certificate_based_authentication, API_Key_based_authentication, and federated_based_authentication. The semantics of the values are:
    - APIGateway: It refers to the use of API Gateway to handle authentication.
    - token_based_authentication: It refers to the use of tokens for authentication.
    - certificate_based_authentication: It refers to the use of certificates for authentication.

        - API_Key_based_authentication: It refers to the use of API keys for authentication.
        - federated_based_authentication: It refers to the use of federated identity solution for authentication.
    - This property can be specified for abstract, system, process, and thread.
    - This property and the property values are derived from the literatures listed in Section 3.1.1. Authentication.
- **Declaration:**

AuthenticationMicroserviceArchitecture: enumeration (APIGateway, token_based_authentication, certificate_based_authentication, API_Key_based_authentication, federated_based_authentication)
applies to (abstract, system, process, thread);

- **Property Association Example:**
    - AuthenticationMicroserviceArchitecture => certificate_based_authentication;

3.2.10. Authorization Related Properties

3.2.10.1. AuthorizationMicroserviceArchitecture

- **Property Name:** AuthorizationMicroserviceArchitecture
- **Description:**
    - This property specifies the authorization mechanism used in the microservice architecture.
    - This property is an enumeration with the values coarse_grained and fine_grained. The semantics of the values are:
        - coarse_grained typically refers to authorization rules that use a single attribute to evaluate a decision to either grant or deny the access, e.g. a particular role has access to a particular resource.

            - fine_grained typically refers to authorization rules that use multiple attributes to evaluate a decision to either grant or deny the access, e.g. a particular role in a particular building has access to a particular resource only during a particular duration of time.
    - This property can be specified for abstract, system, process, and thread.
- **Declaration:**

AuthorizationMicroserviceArchitecture: enumeration (coarse_grained, fine_grained) applies to (abstract, system, process, thread);

- **Property Association Example:**
    - AuthorizationMicroserviceArchitecture => fine_grained;

3.2.10.2. CoarseGrainedArchitecture

- **Property Name:** CoarseGrainedAuthorization
- **Description:**
    - This property specifies the coarse-grained authorization mechanism used in the microservice architecture.
    - This property is an enumeration with the value APIGateway.
    - This property can be specified for abstract, system, process, and thread.
- **Declaration:**

CoarseGrainedAuthorization: enumeration (APIGateway) applies to (abstract, system, process, thread);

- **Property Association Example:**
    - CoarseGrainedAuthorization => APIGateway;

3.2.10.3. FineGrainedArchitecture

- **Property Name:** FineGrainedAuthorization

- **Description:**
    - This property specifies the fine-grained authorization mechanism used in the microservice architecture.
    - This property is an enumeration with the value decentralized and centralized. The semantics of the values are:
        - decentralized: When an access control system is implemented using the decentralized pattern, each microservice is responsible for making access decisions (PDP) and enforcing the access decisions made by the PDP (PEP). This pattern offers more fine-grained access control because the access control rules are more domain specific. However, the development team must be able to configure the access control rules correctly and manual configuration is not scalable.
        - centralized: There are two types of centralized access control: a. Centralized with PDP- Each microservice is responsible for enforcing access control decisions (PEP). The defining of access control rules (PAP), the decision making based on access control rules (PDP), and the maintenance of additional attributes (PIP) are shared among all microservices in the same architecture. b. Centralized with an embedded PDP- Each microservice is responsible for making access decisions (PDP) and enforcing the access decisions made by the PDP (PEP). The access control rules (PAP) and attributes (PIP) are defined centrally and are delivered to embedded PDP in the microservice.
    - This property can be specified for abstract, system, process, and thread.
- **Declaration:**

FineGrainedAuthorization: enumeration (decentralized, centralized) applies to (abstract, system, process, thread);

- **Property Association Example:**

- FineGrainedAuthorization => decentralized;
- FineGrainedAuthorization => centralized;

3.2.10.4. CentralizedFineGrainedAuthorization

- **Property Name:** CentralizedFineGrainedAuthorization
- **Description:**
  - This property specifies the centralized fine grained authorization mechanism used in the microservice architecture.
  - This property is an enumeration with the value withPDP and withEmbeddedPDP. The semantics of the values are:
    - withPDP: Each microservice is responsible for enforcing access control decisions (PEP). The defining of access control rules (PAP), the decision making based on access control rules (PDP), and the maintenance of additional attributes (PIP) are shared among all microservices in the same architecture.
    - withEmbeddedPDP: Each microservice is responsible for making access decisions (PDP) and enforcing the access decisions made by the PDP (PEP). The access control rules (PAP) and attributes (PIP) are defined centrally and are delivered to embedded PDP in the microservice.
  - This property can be specified for abstract, system, process, and thread.
- **Declaration:**

CentralizedFineGrainedAuthorization: enumeration (withPDP, withEmbeddedPDP) applies to (abstract, system, process, thread);

- **Property Association Example:**
  - CentralizedFineGrainedAuthorization => withPDP;
  - CentralizedFineGrainedAuthorization => withEmbeddedPDP;

3.2.11. Decision Trees

Decision trees guide software architects in determining what specific security properties should be considered, how different security properties are related and can be used together, and what additional structural elements (components and connectors) when adding specific security properties. The decision trees associated with the security properties can be found in Appendix A.

Chapter 4: Experiment

This chapter presents an experiment that was conducted to evaluate whether the framework led to an increase in well-justified and articulated security specifications and components in microservice architectures.

4.1. Description of the Experiment

The experiment was designed to assess the hypothesis that use of the framework would lead to an increase in well-justified and articulated security specifications and components in microservice architectures. The independent variable was the use of the decision trees, and the dependent variable was the scores. The null and research hypotheses are as follows:

- Null Hypothesis: Use of the framework does not lead to an increase in well-justified and articulated security specifications and components in microservice architectures.
- Research Hypothesis: Use of the framework leads to a significant increase in well-justified and articulated security specifications and components in microservice architectures.

The participants were undergraduate students from the computer science department at the University of Texas at El Paso. 102 participants completed the "Research Study Background Survey 2023" prior to the experiment. The survey was used to gather information about the participant's software engineering background, such as the number of years of experience in software architecture, software development, and software security. Appendix D presents the result of the research study background survey. Most of the participants have 0 to 1 year of experience in software architecture and software security.

Participants were divided into teams of 3 based on their software engineering background and assigned to either a control group or a treatment group randomly. Based on the T-tests with the alpha level at 5%, power level at 80%, and effect size at 0.866 (for teams of 3), the sample size of the control groups should be 18 groups and the sample size of treatment groups should be 18 groups. Due to the number of participants available on the day of the experiment, there were a total of 15 teams of 3 in the treatment group and 16 teams of 3 in the control group. There were two teams of 4 and one team of 2.

For teams that were assigned to the control group, the following procedures were followed:

- Each team received lecture materials on software architecture with an emphasis on microservice architecture and security.
- Each team received a problem statement, a predefined microservice architecture based on the problem statement, and a survey named "Building Secure Microservice Architecture Survey", and
- Each team was given an hour to complete the survey. Please see Appendix B for the full list of questions included in the survey.

For teams that were assigned to the treatment group, the following procedures were followed:

- Each team received lecture materials on software architecture with an emphasis on microservice architecture and security.
- Each team received a problem statement, a predefined microservice architecture based on the problem statement, and a survey named "Building Secure Microservice Architecture Survey".
- Each team received the framework on how to design secure microservice architecture, and
- Each team was given an hour to complete the survey. Please see Appendix B for the full list of questions included in the survey.

4.1. Evaluation Process

The survey contained a total of 8 questions. Table 1 presents the concepts tested per question. Survey results were analyzed to determine if participants from the treatment groups will have a higher success rate in articulating security specifications and components in microservice architecture using the provided framework than the participants from the control groups.

Table 1: Survey Questions and Concepts Tested

Survey Question Number	**Concepts Tested**
1	a. Trust domain; b. Addition of structural elements (components and connectors) and its associated properties needed to support the trust domain and communication between trust domains; c. Ability to identify the need to annotate the properties listed below on components and connectors; d. Ability to correctly specify the properties listed below on components and connectors; and e. Tested properties: 1. TrustDomain 2. SecurityLevelProvided 3. dbAccessMode 4. DataAtRestEncryption
2	a. Deployment and patching; b. Data; c. Ability to identify the need to annotate the properties listed below on components and connectors; d. Ability to correctly specify the properties listed below on components and connectors; e. Tested properties: 1. DeploymentType 2. patch_type 3. dbAccessMode 4. DataAtRestEncryption.
3	a. Communication; b. Addition of structural structural elements (components and connectors) and its associated properties needed to support the communications between components; c. Ability to identify the need to annotate the properties listed below on components and connectors; d. Ability to correctly specify the properties listed below on components and connectors; and e. Tested properties: 1. SecurityLevelProvided

	2. dbAccessMode 3. DataAtRestEncryption 4. SecureCommunication.
4	a. Edge level security b. Addition of structural elements (components and connectors)and its associated properties needed to support edge security; c. Ability to identify the need to annotate the properties listed below on components and connectors; d. Ability to correctly specify the properties listed below on components and connectors; and e. Tested properties: 1. SecurityLevelProvided 2. EdgeLevel 3. MessagePayloadSizeLimit 4. APIRequestPerSecond_type 5. MaxRateLimitEdge 6. APIRequestPerSecondMicroserviceName_type 7. RequestPerApplicationType 8. APIRequestPerSecondMicroserviceNameApp_type 9. MaxRequestMicroservice.
5	a. Authorization; b. Addition of structural elements (components and connectors)and its associated properties needed to support authorization mechanism;Ability to identify the need to annotate the properties listed below on components and connectors; c. Ability to correctly specify the properties listed below on components and connectors; and d. Tested properties: 1. AuthorizationMicroserviceArchitecture 2. FineGrainedAuthorization 3. CentralizedFineGrainedAuthorization 4. CoarseGrainedAuthorization 5. SecurityLevelProvided 6. dbAccessMode 7. DataAtRestEncryption.
6	a. Data; b. Ability to identify the need to annotate the properties listed below on components and connectors; c. Ability to correctly specify the properties listed below on components and connectors; and d. Tested properties: 1. MicroserviceDataSensitivity.

7	a. Network perimeter level security. b. Addition of structural elements (components and connectors)and its associated properties needed to support network level security; c. Ability to identify the need to annotate the properties listed below on components and connectors; d. Ability to correctly specify the properties listed below on components and connectors; and e. Tested properties: 1. SecurityLevelProvided 2. NetworkPerimeterLevel.
8	a. Authentication. b. Addition of structural elements (components and connectors)and its associated properties needed to support authentication mechanism; c. Ability to identify the need to annotate the properties listed below on components and connectors; d. Ability to correctly specify the properties listed below on components and connectors; and e. Tested properties: 1. AuthenticationMicroserviceArchitecture 2. SecurityLevelProvided 3. dbAccessMode 4. DataAtRestEncryption.

A set of evaluation criteria around identification and specification of components, connectors, and properties was created. Table 2 presents the evaluation criteria used on each question. Table 3 presents the grading scale used in the criteria regarding the identification of properties and structural elements (components and connectors). Table 4 presents the grading scale used in the criteria regarding the specification of properties and structural elements (components and connectors).

Table 2: Evaluation Criteria

Question No.	**Evaluation Criteria**
Question 1	Ability to identify the need to change the trustDomain property for Order microservice
	Ability to specify the trustDomain property for Order microservice
	Ability to identify the need to have a credential microservice in each trust domain

	Ability to identify TrustDomain property in Credential component 1
	Ability to specify TrustDomain property in Credential component 1
	Ability to identify SecurityLevelProvided property in Credential component 1
	Ability to specify SecurityLevelProvided property in Credential component 1
	Ability to identify dbAccessMode property in Credential component 1
	Ability to specify dbAccessMode property in Credential component 1
	Ability to identify DataAtRestEncryption property in Credential component 1
	Ability to specify DataAtRestEncryption property in Credential component 1
Question 2	Ability to identify the need to specify DeploymentType for Account microservice
	Ability to identify the deploymentMechanism field
	Ability to specify the deploymentMechanism field
	Ability to identify the patchList field
	Ability to correct the patchList field
	Ability to identify DataAtRestEncryption for Account DB component needs update
	Ability to specifiy DataAtRestEncryption for Account DB component
Question 3	Ability to identify SecureCommunication property
	Ability to correctly specifiy SecureCommunication property
	Relationship between SecureCommunication and Certificate Authority Component
	Ability to identify the SecurityLevelProvided property
	Ability to correctly specify securityLevelProvided
	Ability to identify dBAccessModel Property
	Ability to identify DataAtRestEncryption property
	Ability to specify dBAccessModel Property
	Ability to specify DataAtRestEncryption property
Question 4	Ability to identify the need to add a new component at the edge to stop direct communication from external applications to microservices
	Ability to identify the securityLeveLProvided for the edge security component
	Ability to correctly specify the SecurityLevelProvided property
	Ability to identify the EdgeLevel property
	Ability to correctly specify the EdgeLevel property
	Ability to identify the MessagePayloadSizeLimit property
	Ability to correctly specify the MessagePayloadSizeLimit property
	Ability to identify the RequestPerApplicationType property

	Ability to correctly specify the RequestPerApplicationType property (microservice_API_requestPerSecondApp)
	Ability to identify the MAXRateLimitEdge property
	Ability to specify the MAXRateLimitEdge property
	Ability to identify the MaxRequestMicroservice Property
	Ability to correctly specify the MaxRequestMicroservice property
Question 5	Ability to identify the need to add PAP
	Ability to identify AuthorizationMicroserviceArchitecture for PAP
	Ability to specify AuthorizationMicroserviceArchitecture for PAP
	Ability to identify FineGrainedAuthorization for PAP
	Ability to specify FineGrainedAuthorization for PAP
	Ability to identify CentralizedAccessControl for PAP
	Ability to specify CentralizedAccessControl for PAP
	Ability to identify SecurityLevelProvided for PAP
	Ability to specify SecurityLevelProvided for PAP
	Ability to identify the need to add PIP
	Ability to identify FineGrainedAuthorization property needs to be updated fr all core microservice
	Ability to specify the FineGrainedAuthorization for all core microservice
	Ability to specify the FineGrainedAuthorization for all core microservice
	Ability to identify CentralizedAccessControl property needs to be updated for all core microservice
	Ability to specify the CentralizedAccessControl for all core microservice
	Ability to identify dbAccessMode for PAP DB
	Ability to specify dbAccessMode for PAP DB
	Ability to identify DataAtRestEncryption for PAP DB
	Ability to specify DataAtRestEncryption for PAP DB
Question 6	Ability to identify MicroserviceDataSensitivity property for Billing Microservice
	Ability to specify dataSensitivityLevel field for Billing Microservice
	Ability to specify microserviceZone field for Billing Microservice
	Ability to identify MicroserviceDataSensitivity property for Payment Microservice
	Ability to specify dataSensitivityLevel field for Payment Microservice
	Ability to specify microserviceZone field for Payment Microservice
	Ability to identify the need to add an intrusion detection system

Question 7	Ability to identify SecurityLevelProvided property
	Ability to specify SecurityLevelProvided property
	Ability to identify NetworkPerimeter property
	Ability to specify NetworkPerimeter property
Question 8	Ability to identify the need to add an authentication service for token generation, issuing, authentication, and invalidation.
	Ability to identify the AuthenticationMicroserviceArchitecture property
	Ability to specify the AuthenticationMicroserviceArchitecture property
	Ability to identify the SecurityLevelProvided property
	Ability to specify the SecurityLevelProvided property
	Ability to identify dBAccessModel property
	Ability to specify dBAccessModel property
	Ability to identify DataAtRestEncryption property
	Ability to specify DataAtRestEncryption property

Table 3: Grading Scale for Criteria Regarding Identification of Properties and Structural Elements

Grade	**Definition**
0	The team did not correctly identify properties, components, or connectors.
5	The team correctly identified properties, components, or connectors

Table 4: Grading Scale for Criteria Regarding Specification of Properties and Structural Elements

Grade	**Definition**
0	The team did not specify properties, components, or connectors.
1	The team was able to specify less than half of the properties, components, or connectors and most of the values are incorrect.
2	The team was able to correctly specify less than half of the properties, components, or connectors.
3	The team was able to correctly specify half of the properties, components or connectors.
4	The team was able to correctly specify more than half of the properties, components, or connectors.

5	The team correctly specified all properties, components, or connectors.

4.3. RESULT OF THE EXPERIMENT

Appendix C presents the scores received by each team for the survey on a per question basis. Table 5 presents the total score received by the treatment groups for the survey on the identification related questions. Table 6 presents the total score received by the control groups for the survey on the identification related questions. Table 7 presents the total score received by the treatment groups for the survey on the specification related questions. Table 8 presents the total score received by the control groups for the survey on the specification related questions.

Table 5: Total Score Received by Treatment Groups for the Survey on Identification Related Questions

Maximum Score Per Question on Identifications	**30**	**20**	**25**	**35**	**50**	**10**	**15**	**25**		**210**
Treatment Group No.	**Q1_I**	**Q2_I**	**Q3_I**	**Q4_I**	**Q5_I**	**Q6_I**	**Q7_I**	**Q8_I**	**Survey Total Score for Identifications for Treatment Group**	**Normalized Total Score for Identifications (%)**
6	30.00	20.00	5.00	30.00	0.00	0.00	15.00	5.00	105.00	50.00
19	0.00	20.00	5.00	25.00	40.00	0.00	0.00	0.00	90.00	42.86
2	5.00	20.00	5.00	20.00	5.00	10.00	15.00	15.00	95.00	45.24
30	5.00	20.00	20.00	10.00	45.00	10.00	15.00	5.00	130.00	61.90
3	5.00	15.00	0.00	20.00	45.00	0.00	15.00	15.00	115.00	54.76
31	5.00	20.00	5.00	10.00	40.00	0.00	0.00	5.00	85.00	40.48
18	30.00	20.00	25.00	30.00	45.00	5.00	0.00	0.00	155.00	73.81
32	30.00	20.00	25.00	15.00	45.00	0.00	15.00	25.00	175.00	83.33
9	5.00	20.00	0.00	10.00	45.00	0.00	5.00	5.00	90.00	42.86
4	15.00	20.00	15.00	25.00	45.00	10.00	15.00	25.00	170.00	80.95
8	5.00	20.00	0.00	10.00	50.00	10.00	5.00	5.00	105.00	50.00
10	5.00	15.00	20.00	25.00	40.00	10.00	15.00	25.00	155.00	73.81

22	5.00	20.00	20.00	0.00	45.00	5.00	15.00	0.00	110.00	52.38
16	5.00	20.00	25.00	30.00	20.00	10.00	15.00	10.00	135.00	64.29
7	20.00	20.00	0.00	0.00	50.00	0.00	5.00	10.00	105.00	50.00
									Average Treatment	**57.78**
									DEV Treatment	**14.36**

Table 6: Total Score Received by Control Groups for the Survey on Identification Related Questions

Maximum Score Per Question on Identifications	**30**	**20**	**25**	**35**	**50**	**10**	**15**	**25**		**210**
Control Group No.	**Q1_I**	**Q2_I**	**Q3_I**	**Q4_I**	**Q5_I**	**Q6_I**	**Q7_I**	**Q8_I**	**Total Score for Identifications for Control Group**	**Normalized Total Score for Identifications (%)**
25	5.00	15.00	5.00	15.00	5.00	0.00	5.00	0.00	50.00	23.81
13	5.00	20.00	10.00	25.00	30.00	0.00	15.00	0.00	105.00	50.00
24	5.00	20.00	5.00	10.00	45.00	0.00	15.00	0.00	100.00	47.62
28	10.00	5.00	0.00	25.00	40.00	0.00	15.00	0.00	95.00	45.24
5	5.00	5.00	0.00	0.00	5.00	0.00	5.00	0.00	20.00	9.52
21	0.00	20.00	5.00	15.00	5.00	0.00	15.00	20.00	80.00	38.10
27	20.00	0.00	5.00	5.00	40.00	0.00	5.00	20.00	95.00	45.24
34	5.00	20.00	10.00	25.00	45.00	5.00	15.00	0.00	125.00	59.52
29	5.00	20.00	5.00	10.00	10.00	0.00	0.00	0.00	50.00	23.81
11	5.00	20.00	0.00	30.00	40.00	5.00	0.00	15.00	115.00	54.76
14	5.00	20.00	10.00	30.00	40.00	0.00	15.00	0.00	120.00	57.14
20	5.00	20.00	0.00	10.00	0.00	0.00	0.00	0.00	35.00	16.67
33	5.00	20.00	15.00	10.00	5.00	10.00	15.00	25.00	105.00	50.00
15	5.00	20.00	0.00	15.00	10.00	0.00	15.00	5.00	70.00	33.33
12	5.00	20.00	5.00	25.00	30.00	5.00	15.00	5.00	110.00	52.38
1	5.00	5.00	0.00	5.00	5.00	0.00	15.00	0.00	35.00	16.67
									Average Control	**38.99**
									DEV Control	**16.19**

Table 7: Total Score Received by Treatment Groups for the Survey on the Specification Related Questions

Maximum Score Per Question on Specifications	**25**	**15**	**20**	**30**	**40**	**20**	**10**	**20**		**180**
Group No.	**Q1_S**	**Q2_S**	**Q3_S**	**Q4_S**	**Q5_S**	**Q6_S**	**Q7_S**	**Q8_S**	**Total Score for Specifications for Treatment Group**	**Normalized Total Score for Specifications (%)**
6	25	15	3	21	0	0	10	5	79	44
19	0	13	5	18	30	0	0	0	66	37
2	5	13	5	10	5	0	10	10	58	32
30	5	13	5	3	10	0	0	5	41	23
3	5	10	0	10	5	0	5	5	40	22
31	5	5	5	0	0	0	0	5	20	11
18	25	15	20	20	35	10	0	0	125	69
32	25	5	20	11	35	0	10	20	126	70
9	0	5	0	3	35	0	0	5	48	27
4	10	15	10	18	35	0	10	20	118	66
8	5	13	0	0	40	10	0	5	73	41
10	5	10	10	18	30	10	10	20	113	63
22	5	15	15	0	35	10	10	0	90	50
16	5	15	20	19	6	12	10	10	97	54
7	5	13	0	0	40	0	5	10	73	41
									Average Treatment	**43.22**
									DEV Treatment	**18.49**

Table 8: Total Score Received by the Control Groups for the Survey on the Specification related Questions

Maximum Score Per Question on Specifications	**25**	**15**	**20**	**30**	**40**	**20**	**10**	**20**		**180**
Group No.	**Q1_S**	**Q2_S**	**Q3_S**	**Q4_S**	**Q5_S**	**Q6_S**	**Q7_S**	**Q8_S**	**Total Score for Specifications for Control Group**	**Normalized Total Score for Specifications (%)**
25	5	10	0	6	5	0	0	0	26	14
13	5	13	5	14	20	0	10	0	67	37
24	5	13	0	1	31	0	10	0	60	33
28	5	5	0	16	26	0	10	0	62	34
5	5	5	0	0	5	0	0	0	15	8
21	0	13	0	5	5	0	10	10	43	24
27	11	0	5	0	35	0	0	11	62	34
34	5	13	5	4	20	10	10	0	67	37
29	5	15	5	6	6	0	0	0	37	21
11	5	13	0	21	30	10	0	6	85	47
14	5	6	0	17	20	0	10	0	58	32
20	5	13	5	5	0	0	0	0	28	16
33	5	13	5	3	5	10	10	20	71	39
15	5	13	0	4	6	0	10	5	43	24
12	5	13	5	12	16	10	10	5	76	42
1	5	5	0	0	5	0	10	5	30	17
									Average Control	**28.82**
									DEV Control	**11.36**

4.4. Analysis of Result

The analysis was done in two parts. The first part was to analyze the participants' ability to identify properties and structural elements. The second part was to analyze the participants' ability to specify properties and structural elements. Figure 8 shows the comparison of total scores for identification of properties and structural elements between treatment and control groups. Assuming this is a normal distribution, 84% of the population in the control group will score less than average of the treatment group. A two-sample t-Test assuming unequal variances for identification was performed. The result of the two-sample t-Test assuming unequal variance for identification, shown in Figure 9, indicates that the difference between the scores for the treatment and control groups are statistically significant, the null hypothesis should be rejected, and the research hypothesis should be accepted.

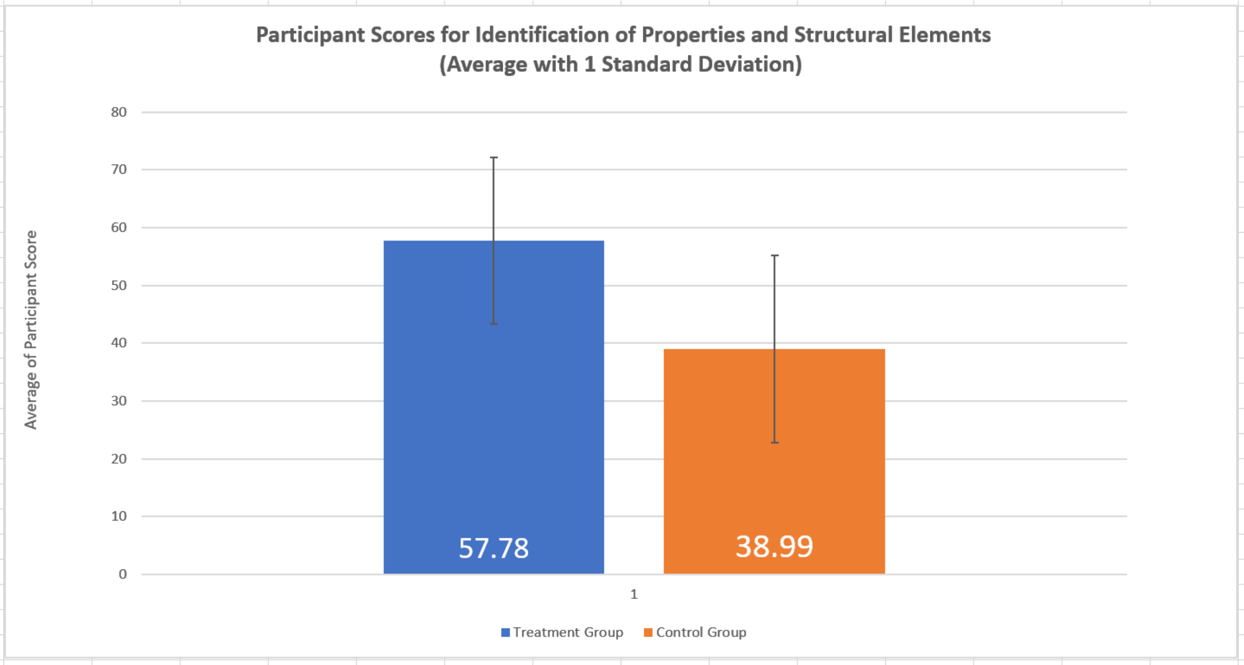
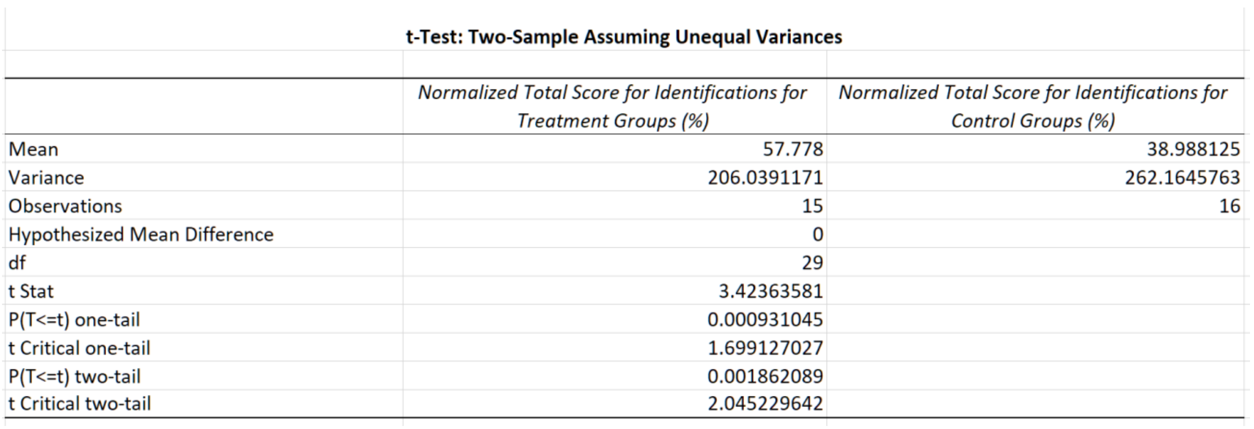


Figure 8:Participant Scores for Identification of Properties and Structural Elements (Average with 1 Standard Deviation)



t-Test: Two-Sample Assuming Unequal Variances

	Normalized Total Score for Identifications for Treatment Groups (%)	*Normalized Total Score for Identifications for Control Groups (%)*
Mean	57.778	38.988125
Variance	206.0391171	262.1645763
Observations	15	16
Hypothesized Mean Difference	0	
df	29	
t Stat	3.42363581	
P(T<=t) one-tail	0.000931045	
t Critical one-tail	1.699127027	
P(T<=t) two-tail	0.001862089	
t Critical two-tail	2.045229642	

Figure 9: Identification: Two Sample t-Test Assuming Unequal Variances

Figure 10 shows the comparison of total scores for specification of properties and structural elements between treatment and control groups. Assuming this is a normal distribution, 84% of the population in the control group will score less than average of the treatment group. A two-sample t-Test assuming unequal variances for specification was performed. The result of the two-sample t-Test assuming unequal variance for specification, shown in Figure 11, indicates that the difference between the scores for the treatment and control groups are statistically significant, the null hypothesis should be rejected, and the research hypothesis should be accepted.

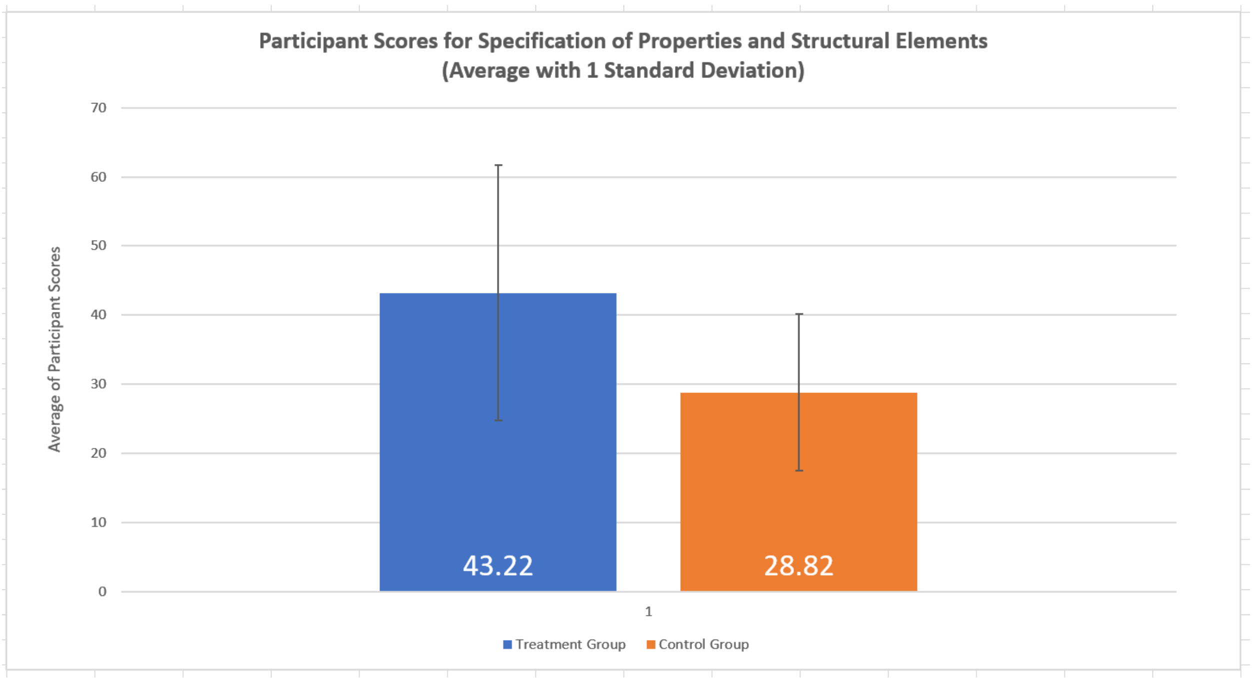
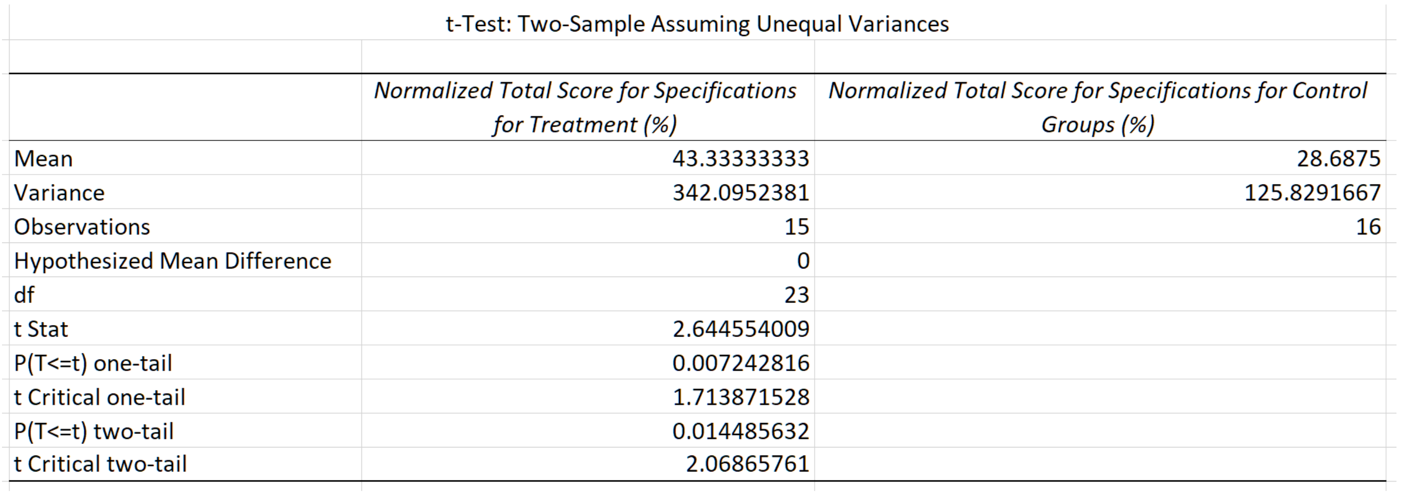


Figure 10: Participant Scores for Specification of Properties and Structural Elements (Average with 1 Standard Deviation)



t-Test: Two-Sample Assuming Unequal Variances

	Normalized Total Score for Specifications for Treatment (%)	*Normalized Total Score for Specifications for Control Groups (%)*
Mean	43.33333333	28.6875
Variance	342.0952381	125.8291667
Observations	15	16
Hypothesized Mean Difference	0	
df	23	
t Stat	2.644554009	
P(T<=t) one-tail	0.007242816	
t Critical one-tail	1.713871528	
P(T<=t) two-tail	0.014485632	
t Critical two-tail	2.06865761	

Figure 11: Specification: Two Sample t-Test Assuming Unequal Variances

4.5. Observations of Result

Four observations can be made after analyzing the data collected:

Observation 1: The participants had difficulty in the identification and specification of properties and structural element (component or connector) when the property has a dependence on a specific structural element.

Observation 2: The participants had difficulty in the identification and specification of properties when the property has a dependence on another property.

Observation 3: The participants had difficulty in the identification and specification of the scope of properties.

Observation 4: The participants had difficulty in the specification of the properties on the appropriate structural element.

Chapter 5: Related Work

5.1. Security Analysis in Software Architecture

This section presents seven approaches for analyzing security in software architecture: Architecture Risk Analysis, Security Vulnerability Approach with SAVE, Attack Surface Security Analysis, Security Architecture Tradeoff Analysis Method, Architecture Analysis for Security, Security Analysis with Acme and Monte Carlo Simulation, and Security Analysis with Information Flow Modeling.

5.1.1. Architecture Risk Analysis

Architectural risk analysis [76,6] (ARA) is a process for identifying flaws in software architecture. It involves examining the required preconditions for vulnerabilities to be exploited and evaluating the potential states the system can be in prior to an exploitation. ARA starts with a one-page architecture that describes the system. The architecture is created by interviewing the software architects, developers, and testers. ARA is guided by three activities: known vulnerability analysis (also known as attack resistance analysis), ambiguity analysis, and underlying platform vulnerability analysis (also known as underlying framework weakness analysis).

Known vulnerability analysis compares the system's architecture against any known attacks, attack patterns, and known principles for confidentiality, integrity, and availability. It assesses the impact of the applicable attacks on the system, identifies vulnerable areas in the architecture, and develops ways to mitigate the risks. Ambiguity analysis aims to eliminate any misunderstandings between requirements and implementation, find weaknesses based on how the system works, and expose any invalid assumptions. It also identifies trust boundaries for function and data (trust modeling); privacy and trust issues for data (data sensitivity modeling); and attackers and areas of weaknesses from the attackers' perspectives (threat modeling). Underlying platform vulnerability analysis examines vulnerabilities associated with the application's execution environment, such as operating system vulnerabilities, network vulnerabilities, platform vulnerabilities, and interaction vulnerabilities resulting from the interaction of components. ARA yields a list of weaknesses associated with the architecture. The analysts rank the weaknesses and propose mitigations.

5.1.2. Security Vulnerability Approach with SAVE

Karppinen, et al. [77] presents the use of Software Architecture Visualization and Evaluation (SAVE) to detect security vulnerabilities that violate structural and behavioral patterns of a software system. SAVE [51] is a non-security specific tool that is used to analyze and optimize the architecture of implemented software systems. It can generate static and dynamic architectural views from source code and compare the architectures for violations. SAVE uses static analysis techniques to reveal dependencies and couplings between components and dynamic analysis techniques to reveal active components in a system's planned and generated architecture design.

Karppinen, et al. uses SAVE to generate static and dynamic views of the implemented architecture from the source code and compare the implemented architecture against the planned architecture. Comparison of the findings in the architectures can determine if the software system is implemented as planned; architecture styles and design patterns are used properly and implemented; and security vulnerabilities and hidden functionalities exist.

This research shows the potential in discovering vulnerabilities in an implemented system by analyzing its architectural design. However, Karppinen, et al. acknowledge that the ability to detect vulnerabilities, attacks, and hidden functionalities is limited because prior knowledge of how the system is attacked is required. The effort in collecting and analyzing such data might not be feasible.

5.1.3. Attack Surface Security Analysis

Gennari and Garlan [33] presents the use of attack surface to evaluate security properties at architectural level and identify architectural vulnerabilities. Attack surface is the measure of a system's exposure to attack. This work is based on Manadahata and Wing's work on attack surfaces. Manadahata and Wing [24] have quantified the attack surface in terms of resources used by a system to interact with its external environment. The resources are entry and exit points, channels, and untrusted data items.

To represent security in an architecture, Gennari and Garlan define the mapping of attack surface to architectural structures. Entry and exit points are mapped to ports. Channels are mapped to an architectural connector that connects components outside of the system with components inside of the system. Untrusted data items are mapped to data sources used by the system that reside in the environment. AcmeStudio, which uses Acme architectural description language, is selected as the architecture modeling environment. For modeling attack surfaces in Acme, Gennari and Garlan defines a new security-focused Acme family and an attack surface plug-in.. The attack surface plug-in allows one to specify attack surface properties of architectural elements. To evaluate security at the architecture, the attack surface plug-in measures the attack surface of an architectural model and identifies principal contributors to the model's attack surface.

5.1.4. Security Architecture Tradeoff Analysis Method

Raza, et al. [78] extends Architecture Tradeoff Analysis Method (ATAM) with security characterization to evaluate security aspects of a software architecture. ATAM is a scenario based method used for analysis of architecture against particular quality goals and how quality goals trade off against each other. ATAM consists of the following phases: presentation, investigation and analysis, testing, and reporting. Presentation phrase involves presenting the ATAM process, the business drivers, and architecture. Investigation and analysis phase involves identifying the architectural approaches used, generating a quality attribute utility tree and scenarios, and analyzing architectural approaches. Testing phase involves brainstorming and prioritizing scenarios and analyzing architectural approaches. Reporting phase involves presenting the results from the analysis of architecture against particular quality goals [54].

Quality attribute characterization helps refine the scenarios created during the investigation and analysis phase. A characterization includes the type of stimuli in which an architecture must respond, the measurable response of the quality attribute by which its achievement is judged, and critical architectural decisions that impact achieving the quality attribute requirement [54]. Raza, et al. create the security characterization. A security stimulus can be a source or type. A source can be an authorized user or unauthorized user. An unauthorized user can be a hacker or attacker. A type can be an internal attack or external attack. An internal attack refers to accidental access to sensitive data. An external attack can be a network attack, data centered attack, application specific attack, or user input attack. A security parameter can be a resource (component) or services. A resource (component) can be a fire wall, virtual lan, proxy server, DMZ, antivirus, certification authority, or operating system. A service can refer to authentication, authorization, access control, intrusion detection, encryption, digital signatures, deception, diversity, or recovery. A security response can be passive or active. A passive response can be protection, prevention, or containment. An active service is a failure recovery. With the security characterization, ATAM can be used to evaluate security aspects of a software architecture following the activities defined in the investigation and analysis phase, testing phase, and reporting phase.

5.1.5. Architectural Analysis for Security

Ryoo, et al. [15] presents the architectural analysis for security (AAFS) method. AAFS consists of three techniques: tactic-oriented architectural analysis (ToAA), pattern-oriented architectural analysis (PoAA), and vulnerability-oriented architectural analysis (VoAA). ToAA involves interviewing an architect about whether and how the system addresses each tactic type and ranking the tactics used in the system to develop a prioritized list of tactics. PoAA involves reviewing the patterns that are related to the identified tactics with the architect, questioning the architect about the existence or use of security patterns and whether the patterns are being implemented correctly. VoAA involves searching for weaknesses resulting from not adopting patterns or not properly implementing the patterns. The output from VoAA's phase is a prioritized list of potential vulnerabilities.

5.1.6. Security Analysis with Acme and Monte Carlo Simulation

Garlan and Schmerl [7, 29] demonstrate how Acme architectural description language along with Monte Carlo simulation can be used to analyze the security of an architecture. A threat scenario includes threat types, assets, and countermeasures (preventative, monitoring, and recovery). Threat type specifies the possible threat that can affect the system. Asset is a component that can be damaged by a threat and is associated with a monetary value. Preventative countermeasure affects the frequency at which a threat occurs. Monitoring countermeasure and recovery countermeasure reduce the effect of a threat. The security simulator in AcmeStudio performs security simulations based on the threat scenarios. The security simulation outputs a report that includes the threat scenario, threat transaction, the most probable damage value to each asset in the threat transaction, and the total damage to the assets in the threat transaction path.

5.1.7. Security Analysis with Information Flow Modeling

Garg, et al. [71] present an approach that uses STRIDE model to define an Acme Data Flow Diagram (DFD) architectural style for security analysis and provide architectural constraints that are used to automatically identify STRIDE threats and security vulnerabilities. The approach starts with an architect modeling a DFD of the system. The architect then specifies properties, such as trust level, for each component(connector). Acme ADL modeling tool checks the DFD against structural and security constraints as defined in the Acme DFD architectural style. The architect will get notified If the Acme ADL modeling tool discovers any threats or vulnerabilities.

5.2. AADL Security Annexes 2019

The AADL security annex 2019 [59] includes the following property sets: security classification property set and security enforcement property set. The security classification property set includes: a. Security clearances (subjects), b. Information security levels (objects), c. Security levels (subjects and objects), and d. Trusted classification. The security enforcement property set includes: a. Data security, b. Data security specification, c. Subject authentication, and d. Secure flows.

The property sets provide in the AADL security annex 2019 covers basic security concepts. It does not have specific security properties that would cover unique security challenges that exist in microservice architectures. The framework described in this dissertation addresses that.

5.3. Summary

A number of approaches on how to analyze security in software architecture is presented. None of the approaches address the problem from a root cause perspective. By knowing the root causes of vulnerabilities, it can assist architects in developing a software architecture that has fewer software weaknesses. Please see Table 9 for the summary comparison table.

Table 9: Summary Comparison Table

Security Approaches	**Formal Specification**	**Modeling Support**	**Threat Modeling**	**Root Cause Analysis**
Architecture Risk Analysis				
Security Vulnerability Approach with SAVE			x	
Attack Surface Security Analysis	x		x	
Security Architecture Tradeoff Analysis Method				
Architectural Analysis for Security				
Security Analysis with Acme and Monte Carlo Simulation	x	Acme	x	
Security Analysis with Information Flow Modeling	x	DFD		
AADL Security Annex 2019	x			
Framework for Security Modeling and Specification in Microservice Architectures	x	AADL		x

Chapter 6: Conclusions

6.1. Summary of Work

There is a lack of consolidated design knowledge on how to build microservice applications. With an increase in the adoption of microservice architecture in the development of applications and the increase in security breaches in microservice based systems, there is a need to examine and understand security issues that exist in microservice architectures. This dissertation presented the Framework for Security Modeling and Specification in Microservice Architectures to enhance the security modeling and specifications in microservice architectures. The research questions that drove the research are:

RQ1: What are the security challenges in microservices architecture?

RQ2: What mechanisms are currently used to address the security challenges in microservices architecture?

RQ3: What approach can enhance the security modeling and specification in microservice architectures?

The outcome was the framework that provides sufficient support in formally defining security properties and adding structural elements in the architecture that address software vulnerabilities in earlier stages of software development of microservice architectures. Please see Table 10 for the mapping of security challenges, practices, properties and decision trees).

An experiment was designed to assess the hypothesis that use of the framework would lead to an increase in well-justified and articulated security specifications and components in microservice architectures. The null and research hypotheses were as follows:

- Null Hypothesis: Use of the framework does not lead to an increase in well-justified and articulated security specifications and components in microservice architectures.
- Research Hypothesis: Use of the framework leads to a significant increase in well-justified and articulated security specifications and components in microservice architectures.

The result of the experiment shows that 84% of the population in the control group will score less than average of the treatment group. A two-sample t-Test assuming unequal variances was performed. The result of the two-sample t-Test assuming unequal variance for identification indicates that the difference between the scores for the treatment and control groups are statistically significant, the null hypothesis should be rejected, and the research hypothesis should be accepted.

This dissertation defines a framework to support the design of microservice architectures and remediate documented security issues. The framework enhances the ability of software architects to formally specify security properties early on in the software development lifecycle. It also includes the use of decision trees to guide software architects in determining what specific security properties should be considered, how different security properties are related and can be used together, and what additional structural elements (components and connectors) should be considered when adding specific security properties. These security properties are derived from existing security challenges and the corresponding security practices used to address them.

The impact of the work is that software vulnerabilities are addressed during early phases of software development (architecture and design) rather than later in the software development lifecycle. This helps to significantly reduce costs associated with software defect mitigation. Studies have shown that the cost ratio in tackling a software defect, including security vulnerabilities, is doubled if defects are discovered during the implementation phase compared to the architecture and design phases. This ratio more than triples if defects are discovered during testing. The work provides comprehensive support in defined security in microservice architectures, especially for software architects who have minimal experience in society.

Table 10: Security Challenges, Practices, Properties, and Decision Tree Summary Table

Security Challenges Category	Security Challenges	Security Practices	Security Properties	Decision Trees
Authentication	Many authentication scenarios compared to an equivalent monolithic architecture and hence increase in complexity in how authentication should be handled.	API Gateway Tokens, such as API token and JSON web token, Certificate-based authentication, API key-based authentication, Hash-based message authentication code, OpenID connect, Federated Identity	SecurityLevelProvided, EdgeLevel, AuthenticationMicroservice Architecture, dbAccessMode, DataAtRest, DataAtRestEncryption,	Edge Level Security Decision Tree, Secure Microservice Architecture Decision Tree, Service Level Decision Tree
	Authentication is a cross cutting concern that affects every microservice, some developers create global authentication logic and assign authentication responsibility to each microservice which is a violation of single responsibility principle.			
	Reusing same code base for authentication creates a central code dependency and negatively impact the technology agnostic aspect of microservices.			
	Management of credentials is challenging since there are more credentials.			
	If authentication information is managed by an authentication microservice, an update is required whenever a new microservice or a new user is added.			
	If the authentication information is managed by individual microservices, it increases the chances of the information being leaked should there be compromises happening			

	to individual microservices.			
Authorization	Many authorization scenarios compared to an equivalent monolithic architecture and hence increase in complexity in how authorization should be handled.	API Gateway, Security Token, OAuth 2.0, Certificates, Access Control System, Decentralized authorization, Centralized Upstream Authorization	AuthorizationMicroservice Architecture, CoarseGrainedAuthorization, FineGrainedAuthorization, CentralizedFineGrainedAuthorization,	Secure Microservice Architecture Decision Tree, Service Level Decision Tree, Edge Level Security Decision Tree
	Authorization is a cross cutting concern that affects every microservice, some developers create global authorization logic and assign authorization responsibility to each microservice which is a violation of single responsibility principle.			
	Reusing same code base for authorization creates a central code dependency and negatively impact the technology agnostic aspect of microservices.			
	Management of credentials and their access rights is challenging since there are more credentials.			
	If a microservice is required to handle authorization at the service level and needs to store and administer user's data, it increases the chances of personal information being leaked and accessed by unauthorized entities.			
	Confused deputy problem refers to an upstream (client) microservice tricks the downstream (server) microservices into doing			

	something they shouldn't be doing.			
	Container-based microservice is immutable meaning that once the container is up, it does not maintain any runtime states or any changes made to its file system. It means that extra steps need to be taken to maintain the dynamic list of allowed clients and access control policies and service credentials since service credentials would be rotated periodically.			
Logging	When microservices are spread across different platforms, security may be out of the control of the microservices owners and completely dependent on the platform environment owner.	Use of distributed tracing system Standard log structure and the amount of information collected	Log	Secure Microservice Architecture Decision Tree, Logging Decision Tree
	Collecting the required and necessary information to diagnose what went wrong and correlating requests among microservices become challenging.			
	For microservices that are deployed using containers, the audit logs are not kept at each node running the microservices.			
Communication	Communication takes place over the network in order to complete requests.	TLS, MTLS	SecureCommunication	Secure Microservice Architecture Decision Tree, Communication Decision Tree
	Improper interception and inappropriate access if teams cannot agree on the communication			

	protocol between microservices.			
Data	Data moves around an architecture more often in a microservice architecture than in a monolithic architecture, and this makes securing data more challenging.	TLS and MTLS, MTLS and Service Mesh, Secure communication protocol, Message authentication code, Encryption, JWT	dbAccessMode, DataAtRest, DataAtRestEncryption, MicroserviceDataSensitivity , SecureCommunication	Secure Microservice Architecture Decision Tree, Data Decision Tree, Communication Decision Tree
	Identity of downstream microservice regarding data in transit and attempt to steal all receiving data.			
	Identity of upstream microservice regarding data in transit and attempt to request for data that it does not have access to.			
	Visibility of data when data is sent across the network.			
	Manipulation of data when data is sent across the network.			
	Data stores in unencrypted form and when an adversary is able to compromise a microservice with an unencrypted data store, he/she will have unlimited access to the data.			
	Amount of data will each microservice needs become questionable since a request is typically fulfilled by more than one microservice.			
Patching	Fail to keep up with patching of vulnerabilities	Assign the right personnel to maintain and handle patching	patch_type, DeploymentType	Secure Microservice Architecture Decision Tree, Deployment and Patching Decision Tree
	Ownership of the infrastructure and software that			

	microservice runs on affects the ability and frequency of patching			
	Dependencies between microservices and third-party libraries used in the development of microservices affect frequency of patching.			
Deployment	Deployment of microservices increases in scale, it makes it extremely challenging to manage and maintain the security	Push or pull model. The service credentials and access control policies are maintained at a policy administration endpoint. With a push model, the policy administration endpoint pushes the updates to the microservice at bootup. With a pull model, the microservice periodically pulls updates from the policy administration endpoint.		
	The technology agnostic nature of microservices also makes vulnerability detection more difficult.			
Trust	Microservices are often designed to trust each other in a microservice architecture. When a malicious adversary attacks and gains control of an individual microservice, it can affect other microservices in the microservice architecture. The malicious adversary can manipulate microservices to do what	Layers of security, Zero trust model, Degree of access separation	TrustDomain, NetworkPerimeterLevel, EdgeLevel, MaxRateLimitEdge, MaxRequestMicroservice, APIRequestPerSecond_type, APIRequestPerSecondMicroserviceName_type, APIRequestPerSecondMicroserviceNameApp_type, APIRequestPerSecondMicroserviceNameUser_type, APIRequestPerSecondMicroserviceNameOperation_ty	Secure Microservice Architecture Decision Tree, Network Perimeter Level Decision Tree, Edge Level Decision Tree, Service Level Decision Tree, Trust Decision Tree

	he/she wants them to do, escalate privileges on the hosting infrastructure of the microservices, listen on any inter-service communication, alter data in transit, lead to full disclosure of other microservices, and potentially bring down the entire system.		pe, RequestPerApplicationType , RequestPerUser, RequestPerOperation, MessagePayloadSizeLimit	
Larger Surface Area	Communications between microservices over the network cause exposure to more potential attacks than a monolithic application due to the increased number in entry points, and hence increases the attack surface area.	API Gateway, Zero Trust Model	EdgeLevel	Secure Microservice Architecture Decision Tree, Edge Level Decision Tree
	With the attack surface area being larger, it makes it harder to manage security.			

6.2. FUTURE WORK

Future work includes enhancing decision trees based on the observations from the experiment (please see section 4.5), automating the support provided by the decision trees, conducting experiments with practitioners, and enhancing the AADL security annex 2019 with the Microservice_Architecture_Security_Properties property set created from this research.

References

1. IDC FutureScape: Worldwide IT Industry 2019 Predictions. (2018). IDC. https://www.idc.com/getdoc.jsp?containerId=US44403818

2. Torkura, K., Sukmana, M., & Meinel, C. (2017). Integrating Continuous Security Assessments in Microservices and Cloud Native Applications. 171–180. https://doi.org/10.1145/3147213.3147229

3. Pereira-Vale, A., Fernandez, E. B., Monge, R., Astudillo, H., & Márquez, G. (2021). Security in microservice-based systems: A Multivocal literature review. 103, 102200. https://doi.org/10.1016/j.cose.2021.102200

4. Mateus-Coelho, N., Cruz-Cunha, M., & Ferreira, L. G. (2021). Security in Microservices Architectures. 181, 1225–1236. https://doi.org/10.1016/j.procs.2021.01.320

5. He, X., & Yang, X. (2017). Authentication and Authorization of End User in Microservice Architecture. 910(1), 12060. https://doi.org/10.1088/1742-6596/910/1/012060

6. McGraw, G. (n.d.). Software Security Touchpoint: Architectural Risk Analysis.

7. Schmerl, B., Gennari, J., & Garlan, D. (2006). Architecture-based Simulation for Security and Performance.

8. Newman, S. (2021). Building Microservices, 2nd Edition. O'Reilly Media, Incorporated. https://learning.oreilly.com/library/view/building-microservices-2nd/9781492034018/

9. Richards, M., & Ford, N. (2020). Fundamentals of Software Architecture. O'Reilly Media, Incorporated. https://learning.oreilly.com/library/view/fundamentals-of-software/9781492043447/

10. Weber, S., Karger, P., & Paradkar, A. (2005). A software flaw taxonomy: Aiming tools at security. 30, 1–7.

11. Baresi, L., & Garriga, M. (2020). Microservices: The Evolution and Extinction of Web Services? (A. Bucchiarone, N. Dragoni, S. Dustdar, P. Lago, M. Mazzara, V. Rivera, & A. Sadovykh (Eds.); pp. 3–28). Springer International Publishing. https://doi.org/10.1007/978-3-030-31646-4_1

12. Kanjilal, J. (2021, November 3). Logging Microservices: The Challenges and Solutions. https://www.developer.com/design/logging-microservices/

13. Ayoub, M. (2018, April 24). Microservices Authentication and Authorization Solutions.

14. An overview of the SSL or TLS handshake. (2023). https://www.ibm.com/docs/en/ibm-mq/7.5?topic=ssl-overview-tls-handshake

15. Ryoo, J., Kazman, R., & Anand, P. (2015). Architectural Analysis for Security (pp. 52–59). IEEE Security & Privacy, vol. 13, no. 6.

16. Parecki, A. (2021, September 2). Hands-on introduction to OAuth 2.0. O'Reilly Media, Incorporated. https://learning.oreilly.com/live-events/hands-on-introduction-to-oauth-20/0636920328384/

17. Powell, O. (2022, November 25). IOTW: Twitter accused of covering up data breach that affects millions.

18. Yarygina, T., & Bagge, A. H. (2018). Overcoming Security Challenges in Microservice Architectures. 11–20. https://doi.org/10.1109/SOSE.2018.00011

19. Henrique, W., Almeida, C., De Aguiar Monteiro, L., Hazin, R. R., Cavalcanti De Lima, A., & Ferraz, F. S. (n.d.). Survey on Microservice Architecture -Security, Privacy and Standardization on Cloud Computing Environment.

20. Lakshminarayanan, S. (2019). AppSecCali 2019 - Authorization in Micro Services World Kubernetes, ISTIO and Open Policy Agent. https://www.youtube.com/watch?v=UnXjwCWgBKU

21. Twitter. (2022, August 5). An incident impacting some accounts and private information on Twitter.

22. Feiler, P., & Gluch, D. (2012). Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley Professional.

23. Berardi, D., Giallorenzo, S., Mauro, J., Melis, A., Montesi, F., & Prandini, M. (2022). Microservice security: a systematic literature review (Vol. 7). PeerJ. https://doi.org/10.7717/peerj-cs.779

24. Manadhata, P., & Wing, J. (2011). An Attack Surface Metric (pp. 371–386). IEEE Trans. Software Eng. 37.

25. Banati, A., Kail, E., Karoczkai, K., & Kozlovszky, M. (2018). Authentication and authorization orchestrator for microservice-based software architectures. In Proceedings of the 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO).

26. Siriwardena, P., & Dias, N. (2022). Microservices Security in Action. Manning Publications Co. https://learning.oreilly.com/library/view/microservices-security-in/9781617295959/

27. Greenberg, A. (2016, August 1). The Jeep Hackers Are Back to Prove Car Hacking Can Get Much Worse.

28. Barabanov, A., & Makrushin, D. (2020). Authentication and Authorization in Microservice-Based Systems: Survey of Architecture Patterns. 4(38), 32–43. https://doi.org/10.21681/2311-3456-2020-04-32-43

29. Garlan, D., & Schmerl, B. (2007). Architecture-driven modelling and analysis (pp. 3–17). Proceedings of the eleventh Australian workshop on Safety critical systems and software - Volume 69 (SCS '06), Tony Cant (Ed.), Vol. 69. Australian Computer Society, Inc.

30. Gough, J., Bryant, D., & Auburn, M. (2022). Mastering API Architecture. O'Reilly Media, Inc. https://learning.oreilly.com/library/view/mastering-api-architecture/9781492090625/

31. HMAC (Hash-Based Message Authentication Codes) Definition. (2023). Okta. https://www.okta.com/identity-101/hmac/

32. Yuqiong, S., Nanda, S., & Jaeger, T. (2015). Security-as-a-Service for Microservices-Based Cloud Applications. 50–57. https://doi.org/10.1109/CloudCom.2015.93

33. Gennari, J., & Garlan, D. (2012). Measuring Attack Surface in Software Architecture (cmu-isr-11-121). CMU.

34. Garlan, D., & Perry, D. (1995). Introduction to the Special Issue on Software Architecture (Vol. 21, Issue 4, pp. 269–274).

35. How SSL and TLS provide identification, authentication, confidentiality, and integrity. (2023). IBM. https://www.ibm.com/docs/en/ibm-mq/7.5?topic=ssl-how-tls-provide-authentication-confidentiality-integrity

36. Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, Today, and Tomorrow (pp. 195–216). Springer International Publishing. https://doi.org/10.1007/978-3-319-67425-4_12

37. Siriwardena, P. (2019). Advanced API Security: OAuth 2.0 and Beyond. Apress. https://learning.oreilly.com/library/view/advanced-api-security/9781484220504/

38. Rountree, D. (2012). Federated Identity Primer. Syngress. https://learning.oreilly.com/library/view/federated-identity-primer/9780124071896/

39. Rehman, S., & Mustafa, K. (2009). Research on software design level security vulnerabilities (pp. 1–5). SIGSOFT Softw. Eng. Notes 34, 6.

40. Santos, J. C. S., Tarrit, K., & Mirakhorli, M. (2017). A Catalog of Security Architecture Weaknesses. 220–223. https://doi.org/10.1109/ICSAW.2017.25

41. Góes de Almeida, M., & Canedo, E. D. (2022). Authentication and Authorization in Microservices Architecture: A Systematic Literature Review. 12(6). https://doi.org/10.3390/app12063023

42. Wilson, Y., & Hingnikar, A. (2022). Solving Identity Management in Modern Applications: Demystifying OAuth 2, OpenID Connect, and SAML 2. O'Reilly Media, Incorporated. https://learning.oreilly.com/library/view/solving-identity-management/9781484282618/

43. IBM Market Development Insights Team, . (2021). Microservices in the enterprise, 2021: Real benefits, worth the challenges. IBM.

44. Gaither, D. (2022). API Keys: API Authentication Methods & Examples. https://blog.stoplight.io/api-keys-best-practices-to-authenticate-apis

45. Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M. (2016). Microservice Architecture. O’Reilly Media, Incorporated. https://learning.oreilly.com/library/view/microservice-architecture/9781491956328/

46. Delange, J. (2016). AADL Security Annex Draft. Software Engineering Institute, Carnegie Mellon University.

47. Venčkauskas, A., Kukta, D., Grigaliūnas, Š., & Brūzgienė, R. (2023). Enhancing Microservices Security with Token-Based Access Control Method. 23(6), 3363. https://doi.org/10.3390/s23063363

48. Security, W. (2018). The Evolution of the Secure Software Lifecycle 2018 Application Security Statistics Report (WhiteHat Security).

49. Lindau, D. (2020, June 23). The Difference Between HTTP Auth, API Keys, and OAuth.

50. DoorDash. (2022, August 25). How we’re responding to a third-party vendor phishing incident.

51. Duszynski, S., Knodel, J., & Lindvall, M. (2009). SAVE: Software Architecture Visualization and Evaluation. 323–324.

52. Office, U. S. G. A. (2018). DATA PROTECTION Actions Taken by Equifax and Federal Agencies in Response to the 2017 Breach Report. United States Government Accountability Office.

53. DoorDash. (2019, September 27). Important security notice about your DoorDash account.

54. Clements, P., Kazman, M., & Klein, M. (2011). Evaluating Software Architectures – Methods and Case Studies. Software Engineering Institute.

55. Richards, M. (2015). Microservices vs. Service-Oriented Architecture. O'Reilly Media, Incorporated. https://learning.oreilly.com/library/view/microservices-vs-service-oriented/9781491975657/cover.html

56. Zettler, K. (n.d.). Platform as a service. Retrieved August 18, 2022, from https://www.atlassian.com/microservices/cloud-computing/platform-as-a-service

57. Subramanian, H., & Raj, P. (2019). Hands-On RESTful API Design Patterns and Best Practices. Packt. https://learning.oreilly.com/library/view/hands-on-restful-api/9781788992664/

58. API Gateway Security. (n.d.). solo.io. https://www.solo.io/topics/api-gateway/api-gateway-security/

59. Gluch, D. (2019). AADL Security Annex. Software Engineering Institute, Carnegie Mellon University.

60. Marvin, M. (2022, August 10). Filling the Access Security Gap With Certificate-Based Authentication. https://www.portnox.com/blog/certificate-based-authentication/

61. Fruhlinger, J. (2020, February 12). Equifax data breach FAQ: What happened, who was affected, what was the impact? https://www.csoonline.com/article/3444488/equifax-data-breach-faq-what-happened-who-was-affected-what-was-the-impact.html

62. Santos, J. C. S. (2016). Toward Establishing a Catalog of Security Architecture Weaknesses", Department of Software Engineering. Rochester Institute of Technology.

63. Fybish, R. (2022, February 3). Authentication in Microservices: Approaches and Techniques.

64. Joseph, C. T., & Chandrasekaran, K. (2019). Straddling the crevasse: A review of microservice software architecture foundations and recent advancements. 49(10), 1448–1484. https://doi.org/10.1002/spe.2729

65. Lewis, J., & Fowler, M. (2014, March 25). Microservices. A definition of this new architectural term .

66. API Gateway: What Is It And Why Is It Essential in Microservices Architecture? (n.d.). traefiklabs. https://traefik.io/glossary/api-gateway-101/

67. Torkura, K. A., Sukmana, M. I. H., Feng Cheng, & Meinel, C. (2017). Leveraging Cloud Native Design Patterns for Security-as-a-Service Applications. 90–97. https://doi.org/10.1109/SmartCloud.2017.21

68. Federated Identity Pattern. (n.d.). Microsoft. https://learn.microsoft.com/en-us/azure/architecture/patterns/federated-identity

69. Symantec. (2016). Internet Security Threat Report. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf

70. Madden, N. (2021). API Security in Action. Manning Publications. https://learning.oreilly.com/library/view/api-security-in/9781617296024/

71. Garg, K., Garlan, D., & Schmerl, B. (2004). Architecture Based Information Flow Analysis for Software Security. Carnegie Mellon University.

72. Penhale, C. (n.d.). Secure Your Container-Based Microservices with Client Certificate Authentication. Openlogic.

73. Santos, J. C. S., Tarrit, K. and Mirakhorli, M. (2017). A Catalog of Security Architecture Weaknesses. IEEE International Conference on Software Architecture Workshops (ICSAW), Gothenburg, pp. 220-223.

74. McGraw, G. (2006). Software Security: Building Security In, Addison-Wesley Professional.

75. Arce, I., Clark-Fisher, K., Daswani, N., DelGrosso, J., Dhillon, D., Kern, C., Kohno, T., Landwehr, C., McGraw, G., Schoenfield, B., Seltzer, M., Spinellis, D., Tarandach, I., West, J. (2014) Avoiding the top 10 software security design flaws, Technical report IEEE Computer Society's Center for Secure Design.

76. Peterson, G., Hope, P., Lavenfar, S. (2005). Architectural Risk Analysis, Cigital, https://www.us-cert.gov/bsi/articles/best-practices/architectural-risk-analysis/architectural-risk-analysis.

77. Karppinen, K., Lindvall, M., Yonkwa, L. (2008). Detecting Security Vulnerabilities with Software Architecture Analysis Tools. IEEE International Conference on Software Testing Verification and Validation Workshop, Lillehammer, Norway.

78. Raza, A., Abbas, H., Yngstrom, L., Hemani, A. (2009). Security characterization for evaluation of software architectures using ATAM. International Conference on Information and Communication Technologies, Karachi, pp. 241-246.

79. Bass, L., Clements, P., Kazman, R. (2012, September). Software Architecture in Practice, Third Edition. Addison-Wesley Professional. https://learning.oreilly.com/library/view/software-architecture-in/9780132942799/

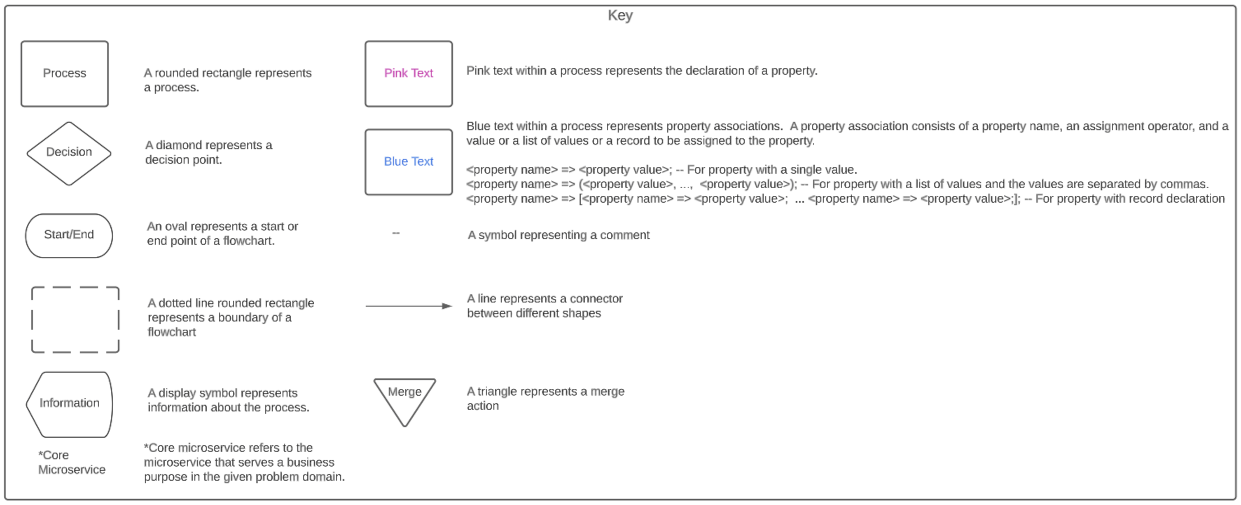
80. Modeling System Architectures using Architecture Analysis and Design Language (AADL). (2015, December). Software Engineering Institute, Carnegie Mellon University.

Appendix A: Decision Trees

Appendix A presents the complete set of decision trees. The decision trees are organized as follows:

1. Key
2. Instructions on How to Use the Decision Trees
3. Secure Microservice Architecture Decision Tree
4. Network Perimeter Level Decision Tree
5. Edge Level Decision Tree Part 1 of 4
6. Edge Level Decision Tree Part 2 of 4
7. Edge Level Decision Tree Part 3 of 4
8. Edge Level Decision Tree Part 4 of 4
9. Service Level Decision Tree Part 1 of 3
10. Service Level Decision Tree Part 2 of 3
11. Service Level Decision Tree Part 3 of 3
12. Communication Decision Tree
13. Logging Decision Tree
14. Deployment and Patching Decision Tree
15. Data Decision Tree
16. Trust Decision Tree.

A.1. KEY



A rounded rectangle represents a process.

A diamond represents a decision point.

An oval represents a start or end point of a flowchart.

A dotted line rounded rectangle represents a boundary of a flowchart

A display symbol represents information about the process.

*Core Microservice

*Core microservice refers to the microservice that serves a business purpose in the given problem domain.

Pink text within a process represents the declaration of a property.

Blue text within a process represents property associations. A property association consists of a property name, an assignment operator, and a value or a list of values or a record to be assigned to the property.

```
<property name> => <property value>; -- For property with a single value.
<property name> => (<property value>, ..., <property value>); -- For property with a list of values and the values are separated by commas.
<property name> => [<property name> => <property value>; ... <property name> => <property value>;]; -- For property with record declaration
```

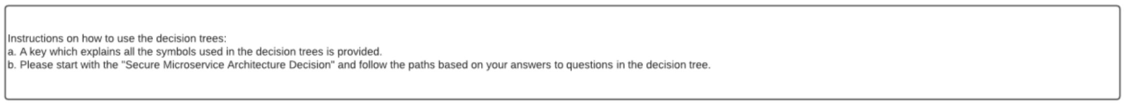
A symbol representing a comment

A line represents a connector between different shapes

A triangle represents a merge action

Figure 12: Decision Tree Key

A.2. Instructions on How to Use the Decision Trees



Instructions on how to use the decision trees:
a. A key which explains all the symbols used in the decision trees is provided.
b. Please start with the "Secure Microservice Architecture Decision" and follow the paths based on your answers to questions in the decision tree.

Figure 13: Instructions on How to Use the Decision Trees

A.3. SECURE MICROSERVICE ARCHITECTURE DECISION TREE

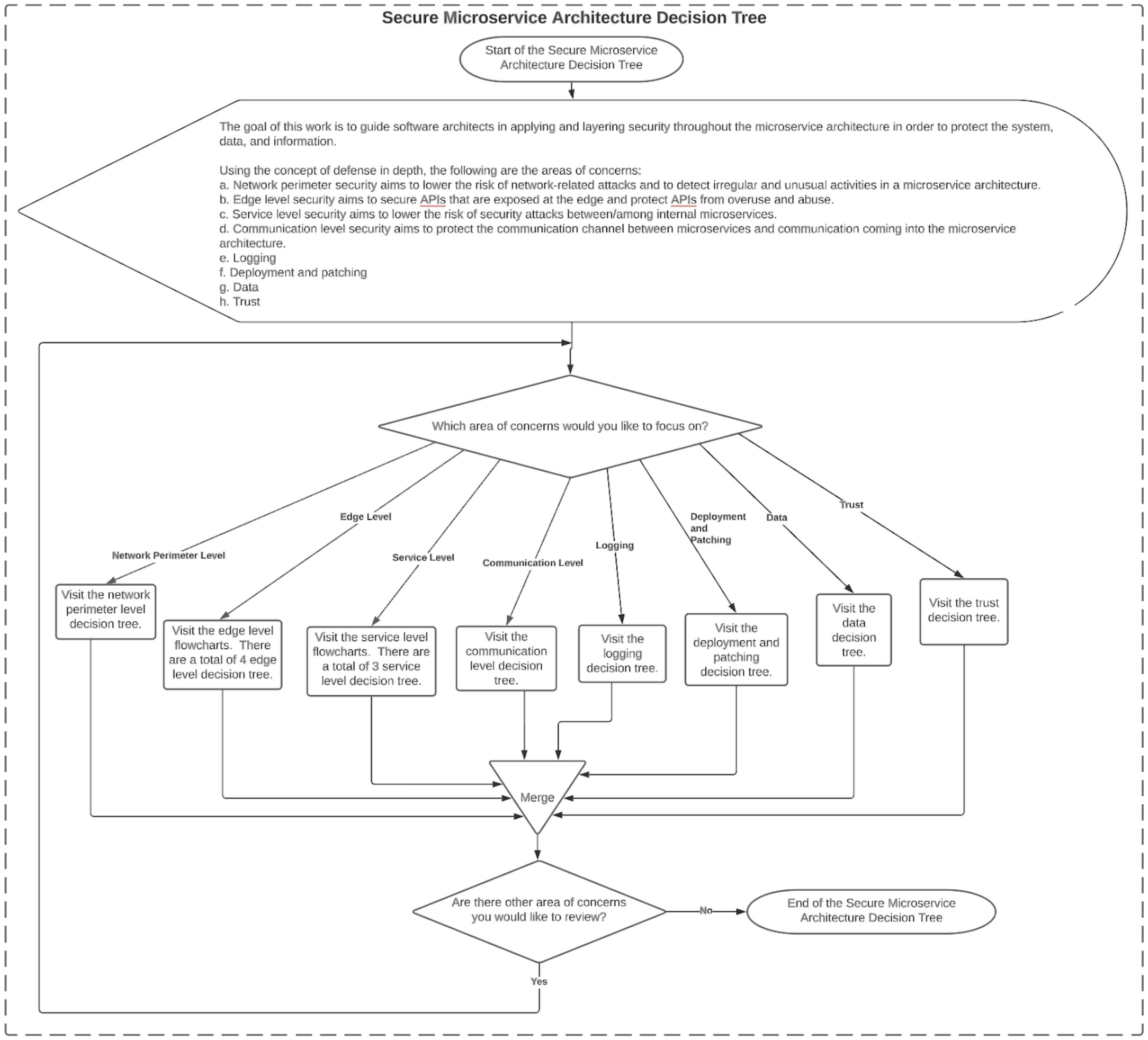
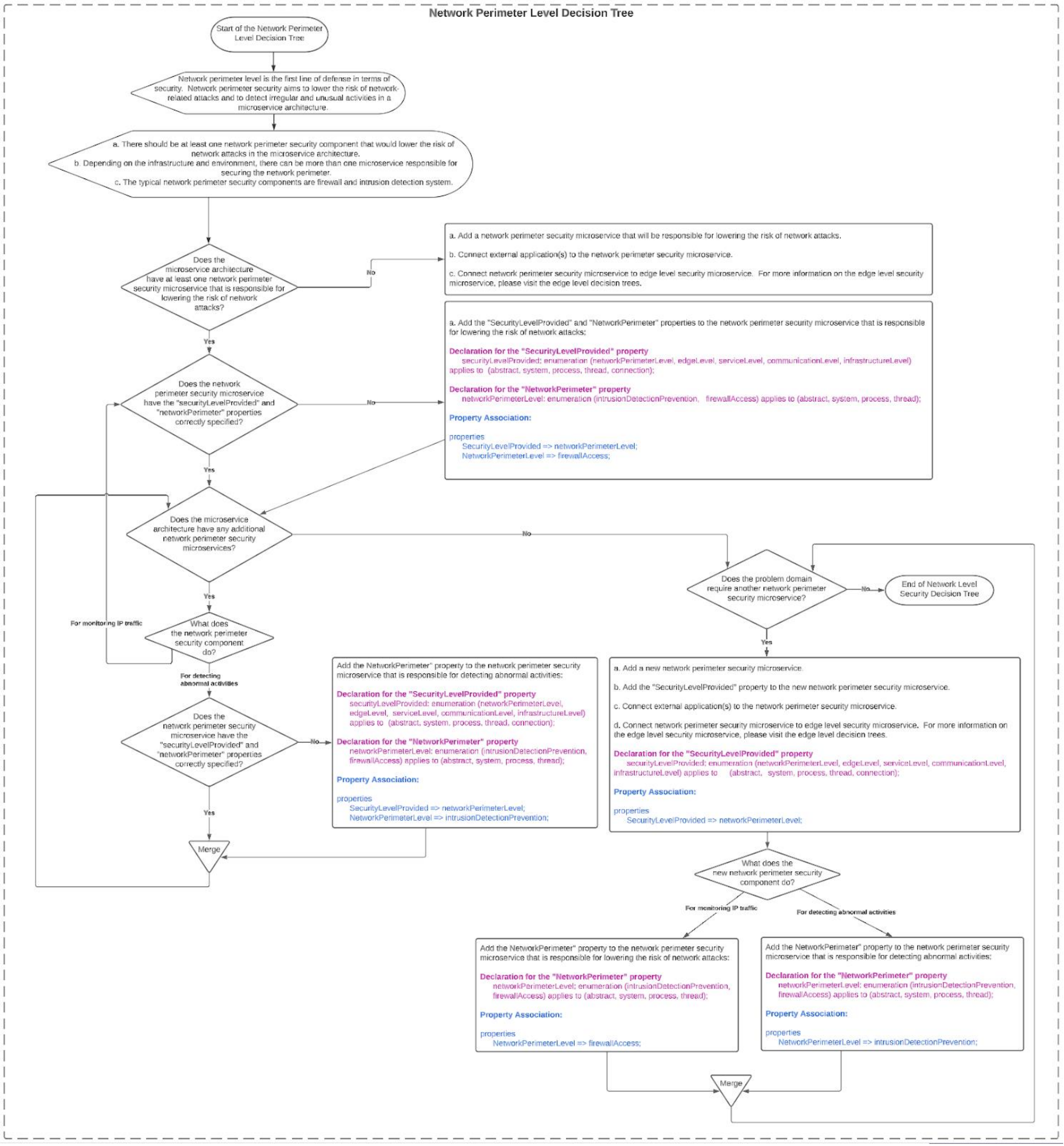

Secure Microservice Architecture Decision Tree
Start of the Secure Microservice Architecture Decision Tree
The goal of this work is to guide software architects in applying and layering security throughout the microservice architecture in order to protect the system, data, and information.
Using the concept of defense in depth, the following are the areas of concerns:
a. Network perimeter security aims to lower the risk of network-related attacks and to detect irregular and unusual activities in a microservice architecture.
b. Edge level security aims to secure APIs that are exposed at the edge and protect APIs from overuse and abuse.
c. Service level security aims to lower the risk of security attacks between/among internal microservices.
d. Communication level security aims to protect the communication channel between microservices and communication coming into the microservice architecture.
e. Logging
f. Deployment and patching
g. Data
h. Trust
Which area of concerns would you like to focus on?
Network Perimeter Level
Edge Level
Service Level
Communication Level
Logging
Deployment and Patching
Data
Trust
Visit the network perimeter level decision tree.
Visit the edge level flowcharts. There are a total of 4 edge level decision tree.
Visit the service level flowcharts. There are a total of 3 service level decision tree.
Visit the communication level decision tree.
Visit the logging decision tree.
Visit the deployment and patching decision tree.
Visit the data decision tree.
Visit the trust decision tree.
Merge
Are there other area of concerns you would like to review?
No
End of the Secure Microservice Architecture Decision Tree
Yes

Figure 14: Secure Microservice Architecture Decision Tree

A.4. Network Perimeter Level Decision Tree


Network Perimeter Level Decision Tree
Start of the Network Perimeter Level Decision Tree
Does the microservice architecture have at least one network perimeter security microservice that is responsible for lowering the risk of network attacks?
No
Yes
Does the network perimeter security microservice have the "securityLevelProvided" and "networkPerimeter" properties correctly specified?
No
Yes
Does the microservice architecture have any additional network perimeter security microservices?
No
Yes
What does the network perimeter security component do?
For monitoring IP traffic
For detecting abnormal activities
Does the network perimeter security microservice have the "securityLevelProvided" and "networkPerimeter" properties correctly specified?
No
Yes
Merge
Does the problem domain require another network perimeter security microservice?
No
End of Network Level Security Decision Tree
Yes
What does the new network perimeter security component do?
For monitoring IP traffic
For detecting abnormal activities
Merge

Network perimeter level is the first line of defense in terms of security. Network perimeter security aims to lower the risk of network-related attacks and to detect irregular and unusual activities in a microservice architecture.

a. There should be at least one network perimeter security component that would lower the risk of network attacks in the microservice architecture.
b. Depending on the infrastructure and environment, there can be more than one microservice responsible for securing the network perimeter.
c. The typical network perimeter security components are firewall and intrusion detection system.

a. Add a network perimeter security microservice that will be responsible for lowering the risk of network attacks.

b. Connect external application(s) to the network perimeter security microservice.

c. Connect network perimeter security microservice to edge level security microservice. For more information on the edge level security microservice, please visit the edge level decision trees.

a. Add the "SecurityLevelProvided" and "NetworkPerimeter" properties to the network perimeter security microservice that is responsible for lowering the risk of network attacks:

Declaration for the "SecurityLevelProvided" property
```
    securityLevelProvided: enumeration (networkPerimeterLevel, edgeLevel, serviceLevel, communicationLevel, infrastructureLevel)
applies to  (abstract, system, process, thread, connection);
```
Declaration for the "NetworkPerimeter" property
```
    networkPerimeterLevel: enumeration (intrusionDetectionPrevention,   firewallAccess) applies to (abstract, system, process, thread);
```
Property Association:
```
properties
    SecurityLevelProvided => networkPerimeterLevel;
    NetworkPerimeterLevel => firewallAccess;
```

Add the NetworkPerimeter" property to the network perimeter security microservice that is responsible for detecting abnormal activities:

Declaration for the "SecurityLevelProvided" property
```
    securityLevelProvided: enumeration (networkPerimeterLevel,
    edgeLevel,  serviceLevel, communicationLevel, infrastructureLevel)
    applies to  (abstract, system, process, thread, connection);
```
Declaration for the "NetworkPerimeter" property
```
    networkPerimeterLevel: enumeration (intrusionDetectionPrevention,
    firewallAccess) applies to (abstract, system, process, thread);
```
Property Association:
```
properties
    SecurityLevelProvided => networkPerimeterLevel;
    NetworkPerimeterLevel => intrusionDetectionPrevention;
```

a. Add a new network perimeter security microservice.

b. Add the "SecurityLevelProvided" property to the new network perimeter security microservice.

c. Connect external application(s) to the network perimeter security microservice.

d. Connect network perimeter security microservice to edge level security microservice. For more information on the edge level security microservice, please visit the edge level decision trees.

Declaration for the "SecurityLevelProvided" property
```
    securityLevelProvided: enumeration (networkPerimeterLevel, edgeLevel, serviceLevel, communicationLevel,
infrastructureLevel) applies to     (abstract,   system, process, thread, connection);
```
Property Association:
```
properties
    SecurityLevelProvided => networkPerimeterLevel;
```

Add the NetworkPerimeter" property to the network perimeter security microservice that is responsible for lowering the risk of network attacks:

Declaration for the "NetworkPerimeter" property
```
    networkPerimeterLevel: enumeration (intrusionDetectionPrevention,
    firewallAccess) applies to (abstract, system, process, thread);
```
Property Association:
```
properties
    NetworkPerimeterLevel => firewallAccess;
```

Add the NetworkPerimeter" property to the network perimeter security microservice that is responsible for detecting abnormal activities:

Declaration for the "NetworkPerimeter" property
```
    networkPerimeterLevel: enumeration (intrusionDetectionPrevention,
    firewallAccess) applies to (abstract, system, process, thread);
```
Property Association:
```
properties
    NetworkPerimeterLevel => intrusionDetectionPrevention;
```

Figure 15: Network Perimeter Level Decision Tree

A.5. Edge Level Perimeter Level Decision Tree Part 1 of 4

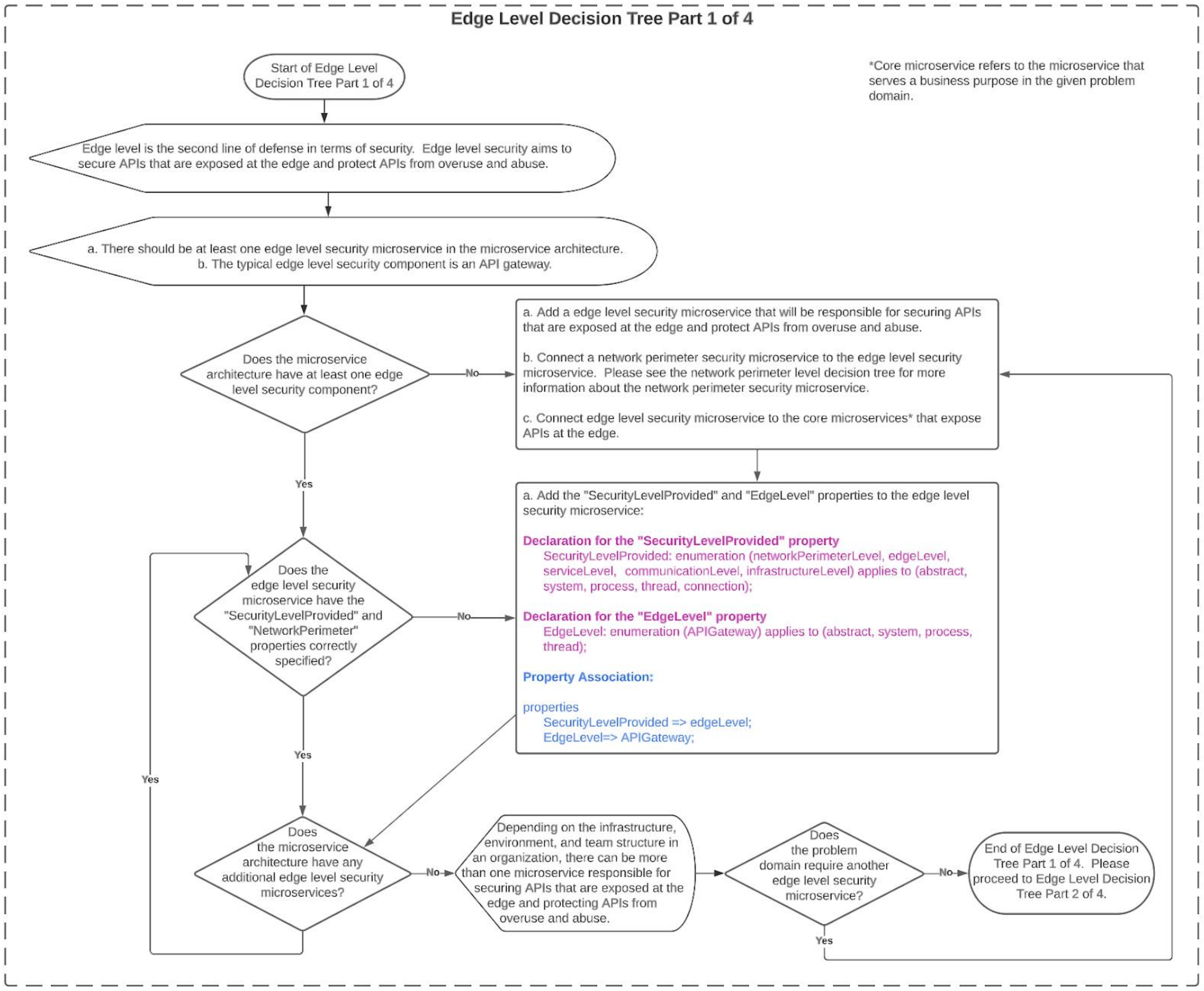
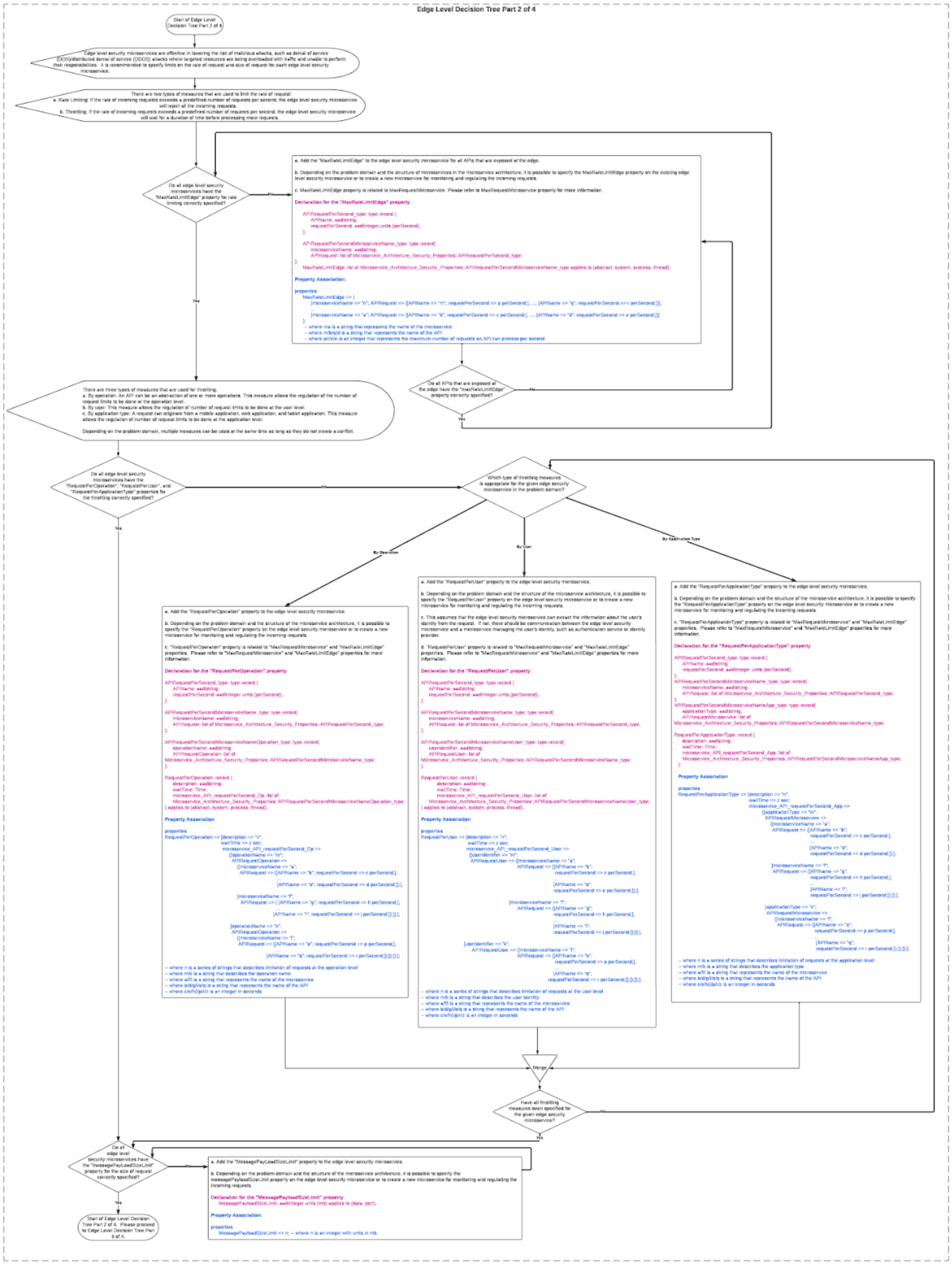

Edge Level Decision Tree Part 1 of 4
Start of Edge Level Decision Tree Part 1 of 4
*Core microservice refers to the microservice that serves a business purpose in the given problem domain.
Edge level is the second line of defense in terms of security. Edge level security aims to secure APIs that are exposed at the edge and protect APIs from overuse and abuse.
a. There should be at least one edge level security microservice in the microservice architecture.
b. The typical edge level security component is an API gateway.
Does the microservice architecture have at least one edge level security component?
No
Yes
a. Add a edge level security microservice that will be responsible for securing APIs that are exposed at the edge and protect APIs from overuse and abuse.
b. Connect a network perimeter security microservice to the edge level security microservice. Please see the network perimeter level decision tree for more information about the network perimeter security microservice.
c. Connect edge level security microservice to the core microservices* that expose APIs at the edge.
Does the edge level security microservice have the "SecurityLevelProvided" and "NetworkPerimeter" properties correctly specified?
No
Yes
Yes
a. Add the "SecurityLevelProvided" and "EdgeLevel" properties to the edge level security microservice:
Declaration for the "SecurityLevelProvided" property
SecurityLevelProvided: enumeration (networkPerimeterLevel, edgeLevel, serviceLevel, communicationLevel, infrastructureLevel) applies to (abstract, system, process, thread, connection);
Declaration for the "EdgeLevel" property
EdgeLevel: enumeration (APIGateway) applies to (abstract, system, process, thread);
Property Association:
properties
SecurityLevelProvided => edgeLevel;
EdgeLevel=> APIGateway;
Does the microservice architecture have any additional edge level security microservices?
No
Depending on the infrastructure, environment, and team structure in an organization, there can be more than one microservice responsible for securing APIs that are exposed at the edge and protecting APIs from overuse and abuse.
Does the problem domain require another edge level security microservice?
No
Yes
End of Edge Level Decision Tree Part 1 of 4. Please proceed to Edge Level Decision Tree Part 2 of 4.

Figure 16:Edge Level Perimeter Level Decision Tree Part 1 of 4

A.6. Edge Level Perimeter Level Decision Tree Part 2 of 4



Edge Level Decision Tree Part 2 of 4
Start of Edge Level Decision Tree Part 2 of 4
Do all edge level security microservices have the "MaxRateLimitEdge" property for rate limiting correctly specified?
No
Yes
Do all APIs that are exposed at the edge have the "maxRateLimitEdge" property correctly specified?
No
Yes
Do all edge level security microservices have the "RequestPerOperation", "RequestPerUser", and "RequestPerApplicationType" properties for the throttling correctly specified?
No
Yes
Which type of throttling measures is appropriate for the given edge security microservice in the problem domain?
By Operation
By User
By Application Type
Merge
Have all throttling measures been specified for the given edge security microservice?
No
Yes
Do all edge level security microservices have the "messagePayLoadSizeLimit" property for the size of request correctly specified?
No
Yes
Start of Edge Level Decision Tree Part 2 of 4. Please proceed to Edge Level Decision Tree Part 3 of 4.

Edge level security microservices are effective in lowering the risk of malicious attacks, such as denial of service (DOS)/distributed denial of service (DDOS) attacks where targeted resources are being overloaded with traffic and unable to perform their responsibilities. It is recommended to specify limits on the rate of request and size of request for each edge level security microservice.

There are two types of measures that are used to limit the rate of request:
a. Rate Limiting: If the rate of incoming requests exceeds a predefined number of requests per second, the edge level security microservice will reject all the incoming requests.
b. Throttling: If the rate of incoming requests exceeds a predefined number of requests per second, the edge level security microservice will wait for a duration of time before processing more requests.

a. Add the "MaxRateLimitEdge" to the edge level security microservice for all APIs that are exposed at the edge.

b. Depending on the problem domain and the structure of microservices in the microservice architecture, it is possible to specify the MaxRateLimitEdge property on the existing edge level security microservice or to create a new microservice for monitoring and regulating the incoming requests.

c. MaxRateLimitEdge property is related to MaxRequestMicroservice. Please refer to MaxRequestMicroservice property for more information.

Declaration for the "MaxRateLimitEdge" property

```
    APIRequestPerSecond_type: type record (
        APIName: aadlstring;
        requestPerSecond: aadlinteger units (perSecond);
    );

    APIRequestPerSecondMicroserviceName_type: type record(
        microserviceName: aadlstring;
        APIRequest: list of Microservice_Architecture_Security_Properties::APIRequestPerSecond_type;
);
    MaxRateLimitEdge: list of Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroserviceName_type applies to (abstract, system, process, thread);
```

Property Association:

```
properties
    MaxRateLimitEdge => (
        [microserviceName => "n"; APIRequest => ([APIName => "m"; requestPerSecond => p perSecond;], ..., [APIName => "q"; requestPerSecond => r perSecond;])],
        ...
        [microserviceName => "a"; APIRequest => ([APIName => "b"; requestPerSecond => c perSecond;], ..., [APIName => "d"; requestPerSecond => e perSecond;])]
    );
     -- where n/a is a string that represents the name of the microservice
     -- where m/b/q/d is a string that represents the name of the API
     -- where p/r/c/e is an integer that represents the maximum number of requests an API can process per second
```

There are three types of measures that are used for throttling:
a. By operation: An API can be an abstraction of one or more operations. This measure allows the regulation of the number of request limits to be done at the operation level.
b. By user: This measure allows the regulation of number of request limits to be done at the user level.
c. By application type: A request can originate from a mobile application, web application, and tablet application. This measure allows the regulation of number of request limits to be done at the application level.

Depending on the problem domain, multiple measures can be used at the same time as long as they do not create a conflict.

a. Add the "RequestPerOperation" property to the edge level security microservice.

b. Depending on the problem domain and the structure of the microservice architecture, it is possible to specify the "RequestPerOperation" property on the edge level security microservice or to create a new microservice for monitoring and regulating the incoming requests.

c. "RequestPerOperation" property is related to "MaxRequestMicroservice" and "MaxRateLimitEdge" properties. Please refer to "MaxRequestMicroservice" and "MaxRateLimitEdge" properties for more information.

Declaration for the "RequestPerOperation" property

```
APIRequestPerSecond_type: type record (
    APIName: aadlstring;
    requestPerSecond: aadlinteger units (perSecond);
);

APIRequestPerSecondMicroserviceName_type: type record(
    microserviceName: aadlstring;
    APIRequest: list of Microservice_Architecture_Security_Properties::APIRequestPerSecond_type;
);

APIRequestPerSecondMicroserviceNameOperation_type: type record(
    operationName: aadlstring;
    APIRequestOperation: list of
Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroserviceName_type;
);

RequestPerOperation: record (
    description: aadlstring;
    waitTime: Time;
    microservice_API_requestPerSecond_Op: list of
    Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroserviceNameOperation_type;
) applies to (abstract, system, process, thread);
```

Property Association

```
properties
RequestPerOperation => [description => "n";
                        waitTime => z sec;
                         microservice_API_requestPerSecond_Op =>
                            ([operationName => "m";
                             APIRequestOperation =>
                               ([microserviceName => "a";
                                 APIRequest => ([APIName => "b"; requestPerSecond => c perSecond;],
                                               ...
                                               [APIName => "d"; requestPerSecond => d perSecond;]);],

                               [microserviceName => "f";
                                 APIRequest => ( [APIName => "g"; requestPerSecond => h perSecond;],
                                               ...
                                              [APIName => "i"; requestPerSecond => j perSecond;]);]);],

                            [operationName => "k";
                             APIRequestOperation =>
                               ([microserviceName => "l";
                                APIRequest => ([APIName => "o"; requestPerSecond => p perSecond;],
                                               ...
                                            [APIName => "q"; requestPerSecond => r perSecond;]);]);]);];

-- where n is a series of strings that describes limitation of requests at the operation level
-- where m/k is a string that describes the operation name
-- where a/f/l is a string that represents the name of the microservice
-- where b/d/g/i/o/q is a string that represents the name of the API
-- where c/e/h/j/p/r/z is an integer in seconds
```

a. Add the "RequestPerUser" property to the edge level security microservice.

b. Depending on the problem domain and the structure of the microservice architecture, it is possible to specify the "RequestPerUser" property on the edge level security microservice or to create a new microservice for monitoring and regulating the incoming requests.

c. This assumes that the edge level security microservice can extract the information about the user's identity from the request. If not, there should be communication between the edge level security microservice and a microservice managing the user's identity, such as authentication service or identity provider.

d. "RequestPerUser" property is related to "MaxRequestMicroservice" and "MaxRateLimitEdge" properties. Please refer to "MaxRequestMicroservice" and "MaxRateLimitEdge" properties for more information.

Declaration for the "RequestPerUser" property

```
APIRequestPerSecond_type: type record (
    APIName: aadlstring;
    requestPerSecond: aadlinteger units (perSecond);
);

APIRequestPerSecondMicroserviceName_type: type record(
    microserviceName: aadlstring;
    APIRequest: list of Microservice_Architecture_Security_Properties::APIRequestPerSecond_type;
);

APIRequestPerSecondMicroserviceNameUser_type: type record(
    userIdentifier: aadlstring;
    APIRequestUser: list of
Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroserviceName_type;
);

RequestPerUser: record (
        description: aadlstring;
        waitTime: Time;
        microservice_API_requestPerSecond_User: list of
    Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroserviceNameUser_type;
) applies to (abstract, system, process, thread);
```

Property Association

```
properties
RequestPerUser => [description => "n";
                    waitTime => z sec;
                   microservice_API_requestPerSecond_User =>
                    ([userIdentifier => "m";
                     APIRequestUser => ([microserviceName => "a";
                                         APIRequest => ([APIName => "b";
                                                         requestPerSecond => c perSecond;],
                                                        ...
                                                        [APIName => "d";
                                                        requestPerSecond => e perSecond;]);],

                                       [microserviceName => "f";
                                         APIRequest => ([APIName => "g";
                                                        requestPerSecond => h perSecond;],
                                                        ...
                                                        [APIName => "i";
                                                        requestPerSecond => j perSecond;]);]);],

                  [userIdentifier => "k";
                     APIRequestUser => ([microserviceName => "l";
                                         APIRequest => ([APIName => "o";
                                                         requestPerSecond => p perSecond;],
                                                        ...
                                                        [APIName => "q";
                                                      requestPerSecond => r perSecond;]);]);]);];

-- where n is a series of strings that describes limitation of requests at the user level
-- where m/k is a string that describes the user identity
-- where a/f/l is a string that represents the name of the microservice
-- where b/d/g/i/o/q is a string that represents the name of the API
-- where c/e/h/j/p/r/z is an integer in seconds
```

a. Add the "RequestPerApplicationType" property to the edge level security microservice.

b. Depending on the problem domain and the structure of the microservice architecture, it is possible to specify the "RequestPerApplicationType" property on the edge level security microservice or to create a new microservice for monitoring and regulating the incoming requests.

c. "RequestPerApplicationType" property is related to "MaxRequestMicroservice" and "MaxRateLimitEdge" properties. Please refer to "MaxRequestMicroservice" and "MaxRateLimitEdge" properties for more information.

Declaration for the "RequestPerApplicationType" property

```
APIRequestPerSecond_type: type record (
    APIName: aadlstring;
    requestPerSecond: aadlinteger units (perSecond);
);
APIRequestPerSecondMicroserviceName_type: type record(
    microserviceName: aadlstring;
    APIRequest: list of Microservice_Architecture_Security_Properties::APIRequestPerSecond_type;
);
APIRequestPerSecondMicroserviceNameApp_type: type record(
    applicationType: aadlstring;
    APIRequestMicroservice: list of
Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroserviceName_type;

RequestPerApplicationType: record (
    description: aadlstring;
    waitTime: Time;
    microservice_API_requestPerSecond_App: list of
    Microservice_Architecture_Security_Properties::APIRequestPerSecondMicroserviceNameApp_type;
  );
```

Property Association

```
properties
RequestPerApplicationType => [description => "n";
                              waitTime => z sec;
                              microservice_API_requestPerSecond_App =>
                                 ([applicationType => "m";
                                  APIRequestMicroservice =>
                                    ([microserviceName => "a";
                                     APIRequest => ([APIName => "b";
                                                    requestPerSecond => c perSecond;],
                                                    ...
                                                   [APIName => "d";
                                                    requestPerSecond => d perSecond;]);],

                                    [microserviceName => "f";
                                     APIRequest => ([APIName => "g";
                                                    requestPerSecond => h perSecond;],
                                                    ...
                                                    [APIName => "i";
                                                    requestPerSecond => j perSecond;]);]);],

                                 [applicationType => "k";
                                  APIRequestMicroservice =>
                                     ([microserviceName => "l";
                                      APIRequest => ([APIName => "o";
                                                     requestPerSecond => p perSecond;],
                                                     ...
                                                     [APIName => "q";
                                                     requestPerSecond => r perSecond;]);]);]);];

 -- where n is a series of strings that describes limitation of requests at the application level
 -- where m/k is a string that describes the application type
 -- where a/f/l is a string that represents the name of the microservice
 -- where b/d/g/i/o/q is a string that represents the name of the API
 -- where c/e/h/j/p/r/z is an integer in seconds
```

a. Add the "MessagePayLoadSizeLimit" property to the edge level security microservice.

b. Depending on the problem domain and the structure of the microservice architecture, it is possible to specify the messagePayloadSizeLimit property on the edge level security microservice or to create a new microservice for monitoring and regulating the incoming requests.

Declaration for the "MessagePayloadSizeLimit" property

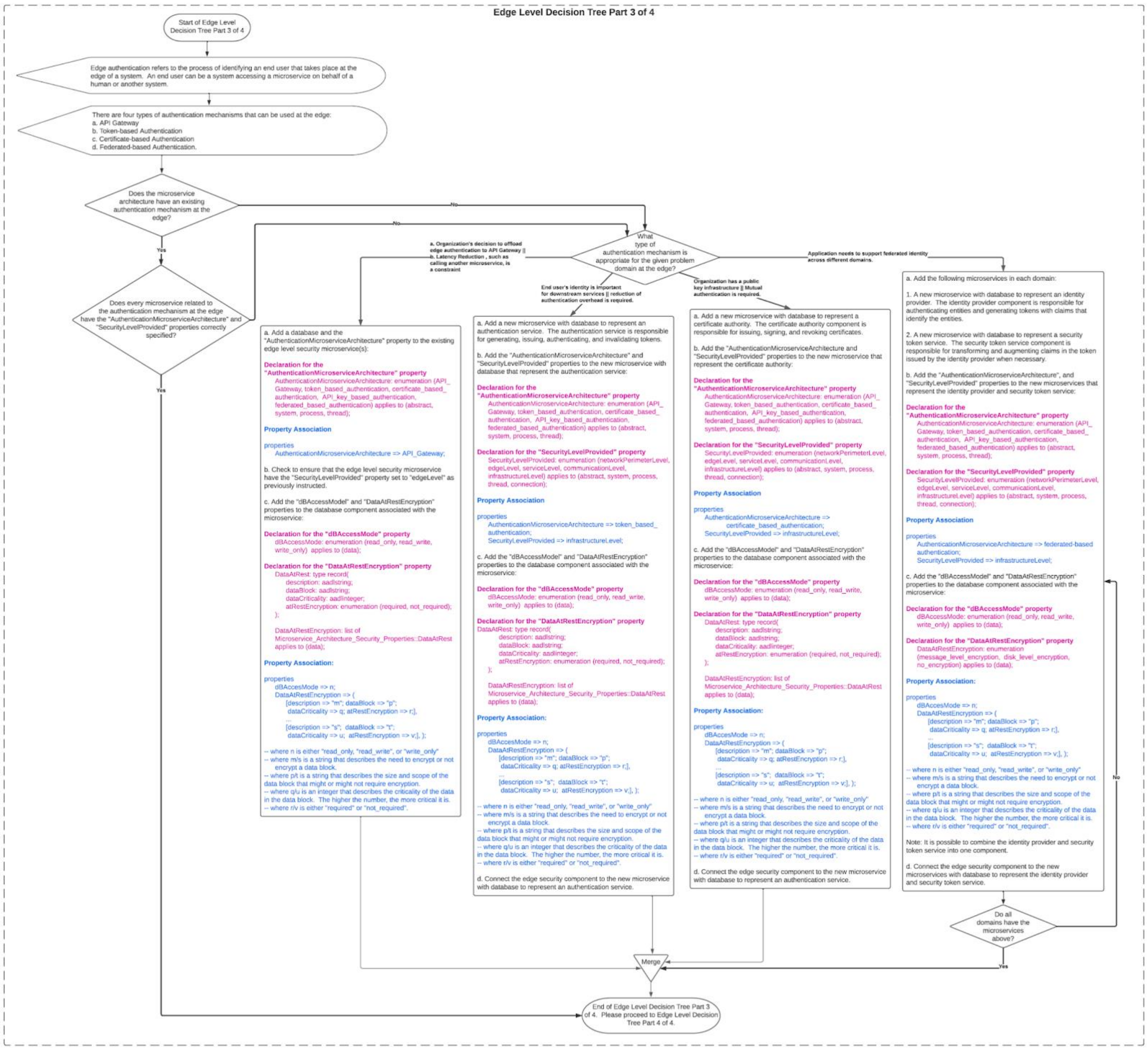
```
    MessagePayloadSizeLimit: aadlinteger units (mb) applies to (data, port);
```

Property Association:

```
properties
    MessagePayloadSizeLimit => n; -- where n is an integer with units in mb.
```

Figure 17: Edge Level Perimeter Level Decision Tree Part 2 of 4

A.7. Edge Level Perimeter Level Decision Tree Part 3 of 4


Edge Level Decision Tree Part 3 of 4
Start of Edge Level Decision Tree Part 3 of 4
Does the microservice architecture have an existing authentication mechanism at the edge?
No
Yes
Does every microservice related to the authentication mechanism at the edge have the "AuthenticationMicroserviceArchitecture" and "SecurityLevelProvided" properties correctly specified?
No
Yes
What type of authentication mechanism is appropriate for the given problem domain at the edge?
a. Organization's decision to offload edge authentication to API Gateway || b. Latency Reduction , such as calling another microservice, is a constraint
End user's identity is important for downstream services || reduction of authentication overhead is required.
Organization has a public key infrastructure || Mutual authentication is required.
Application needs to support federated identity across different domains.
Do all domains have the microservices above?
No
Yes
Merge
End of Edge Level Decision Tree Part 3 of 4. Please proceed to Edge Level Decision Tree Part 4 of 4.

Edge authentication refers to the process of identifying an end user that takes place at the edge of a system. An end user can be a system accessing a microservice on behalf of a human or another system.

There are four types of authentication mechanisms that can be used at the edge:
a. API Gateway
b. Token-based Authentication
c. Certificate-based Authentication
d. Federated-based Authentication.

a. Add a database and the "AuthenticationMicroserviceArchitecture" property to the existing edge level security microservice(s):

```
Declaration for the
"AuthenticationMicroserviceArchitecture" property
    AuthenticationMicroserviceArchitecture: enumeration (API_
    Gateway, token_based_authentication, certificate_based_
    authentication,  API_key_based_authentication,
    federated_based_authentication) applies to (abstract,
    system, process, thread);

Property Association

properties
    AuthenticationMicroserviceArchitecture => API_Gateway;
```

b. Check to ensure that the edge level security microservice have the "SecurityLevelProvided" property set to "edgeLevel" as previously instructed.

c. Add the "dBAccessModel" and "DataAtRestEncryption" properties to the database component associated with the microservice:

```
Declaration for the "dBAccessMode" property
    dBAccessMode: enumeration (read_only, read_write,
    write_only)  applies to (data);

Declaration for the "DataAtRestEncryption" property
    DataAtRest: type record(
        description: aadlstring;
        dataBlock: aadlstring;
        dataCriticality: aadlinteger;
        atRestEncryption: enumeration (required, not_required);
    );

    DataAtRestEncryption: list of
    Microservice_Architecture_Security_Properties::DataAtRest
    applies to (data);

Property Association:

properties
    dBAccesMode => n;
    DataAtRestEncryption => (
        [description => "m"; dataBlock => "p";
         dataCriticality => q; atRestEncryption => r;],
        ...
        [description => "s";  dataBlock => "t";
         dataCriticality => u;  atRestEncryption => v;], );

-- where n is either "read_only, "read_write", or "write_only"
-- where m/s is a string that describes the need to encrypt or not
   encrypt a data block.
-- where p/t is a string that describes the size and scope of the
data block that might or might not require encryption.
-- where q/u is an integer that describes the criticality of the data
in the data block.  The higher the number, the more critical it is.
-- where r/v is either "required" or "not_required".
```

a. Add a new microservice with database to represent an authentication service. The authentication service is responsible for generating, issuing, authenticating, and invalidating tokens.

b. Add the "AuthenticationMicroserviceArchitecture" and "SecurityLevelProvided" properties to the new microservice with database that represent the authentication service:

```
Declaration for the
"AuthenticationMicroserviceArchitecture" property
    AuthenticationMicroserviceArchitecture: enumeration (API_
    Gateway, token_based_authentication, certificate_based_
    authentication,  API_key_based_authentication,
    federated_based_authentication) applies to (abstract,
    system, process, thread);

Declaration for the "SecurityLevelProvided" property
    SecurityLevelProvided: enumeration (networkPerimeterLevel,
    edgeLevel, serviceLevel, communicationLevel,
    infrastructureLevel) applies to (abstract, system, process,
    thread, connection);

Property Association

properties
    AuthenticationMicroserviceArchitecture => token_based_
    authentication;
    SecurityLevelProvided => infrastructureLevel;
```

c. Add the "dBAccessModel" and "DataAtRestEncryption" properties to the database component associated with the microservice:

```
Declaration for the "dBAccessMode" property
    dBAccessMode: enumeration (read_only, read_write,
    write_only)  applies to (data);

Declaration for the "DataAtRestEncryption" property
DataAtRest: type record(
        description: aadlstring;
        dataBlock: aadlstring;
        dataCriticality: aadlinteger;
        atRestEncryption: enumeration (required, not_required);
    );

    DataAtRestEncryption: list of
    Microservice_Architecture_Security_Properties::DataAtRest
    applies to (data);

Property Association:

properties
    dBAccesMode => n;
    DataAtRestEncryption => (
        [description => "m"; dataBlock => "p";
         dataCriticality => q; atRestEncryption => r;],
        ...
        [description => "s";  dataBlock => "t";
         dataCriticality => u;  atRestEncryption => v;], );

-- where n is either "read_only, "read_write", or "write_only"
-- where m/s is a string that describes the need to encrypt or not
   encrypt a data block.
-- where p/t is a string that describes the size and scope of the
data block that might or might not require encryption.
-- where q/u is an integer that describes the criticality of the data
in the data block.  The higher the number, the more critical it is.
-- where r/v is either "required" or "not_required".
```

d. Connect the edge security component to the new microservice with database to represent an authentication service.

a. Add a new microservice with database to represent a certificate authority. The certificate authority component is responsible for issuing, signing, and revoking certificates.

b. Add the "AuthenticationMicroserviceArchitecture and "SecurityLevelProvided" properties to the new microservice that represent the certificate authority:

```
Declaration for the
"AuthenticationMicroserviceArchitecture" property
    AuthenticationMicroserviceArchitecture: enumeration (API_
    Gateway, token_based_authentication, certificate_based_
    authentication,  API_key_based_authentication,
    federated_based_authentication) applies to (abstract,
    system, process, thread);

Declaration for the "SecurityLevelProvided" property
    SecurityLevelProvided: enumeration (networkPerimeterLevel,
    edgeLevel, serviceLevel, communicationLevel,
    infrastructureLevel) applies to (abstract, system, process,
    thread, connection);

Property Association

properties
    AuthenticationMicroserviceArchitecture =>
            certificate_based_authentication;
    SecurityLevelProvided => infrastructureLevel;
```

c. Add the "dBAccessModel" and "DataAtRestEncryption" properties to the database component associated with the microservice:

```
Declaration for the "dBAccessMode" property
    dBAccessMode: enumeration (read_only, read_write,
    write_only)  applies to (data);

Declaration for the "DataAtRestEncryption" property
    DataAtRest: type record(
        description: aadlstring;
        dataBlock: aadlstring;
        dataCriticality: aadlinteger;
        atRestEncryption: enumeration (required, not_required);
    );

    DataAtRestEncryption: list of
    Microservice_Architecture_Security_Properties::DataAtRest
    applies to (data);

Property Association:

properties
    dBAccesMode => n;
    DataAtRestEncryption => (
        [description => "m"; dataBlock => "p";
         dataCriticality => q; atRestEncryption => r;],
        ...
        [description => "s";  dataBlock => "t";
         dataCriticality => u;  atRestEncryption => v;], );

-- where n is either "read_only, "read_write", or "write_only"
-- where m/s is a string that describes the need to encrypt or not
   encrypt a data block.
-- where p/t is a string that describes the size and scope of the
data block that might or might not require encryption.
-- where q/u is an integer that describes the criticality of the data
in the data block.  The higher the number, the more critical it is.
-- where r/v is either "required" or "not_required".
```

d. Connect the edge security component to the new microservice with database to represent an authentication service.

a. Add the following microservices in each domain:

1. A new microservice with database to represent an identity provider. The identity provider component is responsible for authenticating entities and generating tokens with claims that identify the entities.

2. A new microservice with database to represent a security token service. The security token service component is responsible for transforming and augmenting claims in the token issued by the identity provider when necessary.

b. Add the "AuthenticationMicroserviceArchitecture", and "SecurityLevelProvided" properties to the new microservices that represent the identity provider and security token service:

```
Declaration for the
"AuthenticationMicroserviceArchitecture" property
    AuthenticationMicroserviceArchitecture: enumeration (API_
    Gateway, token_based_authentication, certificate_based_
    authentication,  API_key_based_authentication,
    federated_based_authentication) applies to (abstract,
    system, process, thread);

Declaration for the "SecurityLevelProvided" property
    SecurityLevelProvided: enumeration (networkPerimeterLevel,
    edgeLevel, serviceLevel, communicationLevel,
    infrastructureLevel) applies to (abstract, system, process,
    thread, connection);

Property Association

properties
    AuthenticationMicroserviceArchitecture => federated-based
    authentication;
    SecurityLevelProvided => infrastructureLevel;
```

c. Add the "dBAccessModel" and "DataAtRestEncryption" properties to the database component associated with the microservice:

```
Declaration for the "dBAccessMode" property
    dBAccessMode: enumeration (read_only, read_write,
    write_only)  applies to (data);

Declaration for the "DataAtRestEncryption" property
    DataAtRestEncryption: enumeration
    (message_level_encryption,  disk_level_encryption,
    no_encryption) applies to (data);

Property Association:

properties
    dBAccesMode => n;
    DataAtRestEncryption => (
        [description => "m"; dataBlock => "p";
         dataCriticality => q; atRestEncryption => r;],
        ...
        [description => "s";  dataBlock => "t";
         dataCriticality => u;  atRestEncryption => v;], );

-- where n is either "read_only, "read_write", or "write_only"
-- where m/s is a string that describes the need to encrypt or not
   encrypt a data block.
-- where p/t is a string that describes the size and scope of the
data block that might or might not require encryption.
-- where q/u is an integer that describes the criticality of the data
in the data block.  The higher the number, the more critical it is.
-- where r/v is either "required" or "not_required".
```

Note: It is possible to combine the identity provider and security token service into one component.

d. Connect the edge security component to the new microservices with database to represent the identity provider and security token service.

Figure 18: Edge Level Perimeter Level Decision Tree Part 3 of 4

A.8. EDGE LEVEL PERIMETER LEVEL DECISION TREE PART 4 OF 4

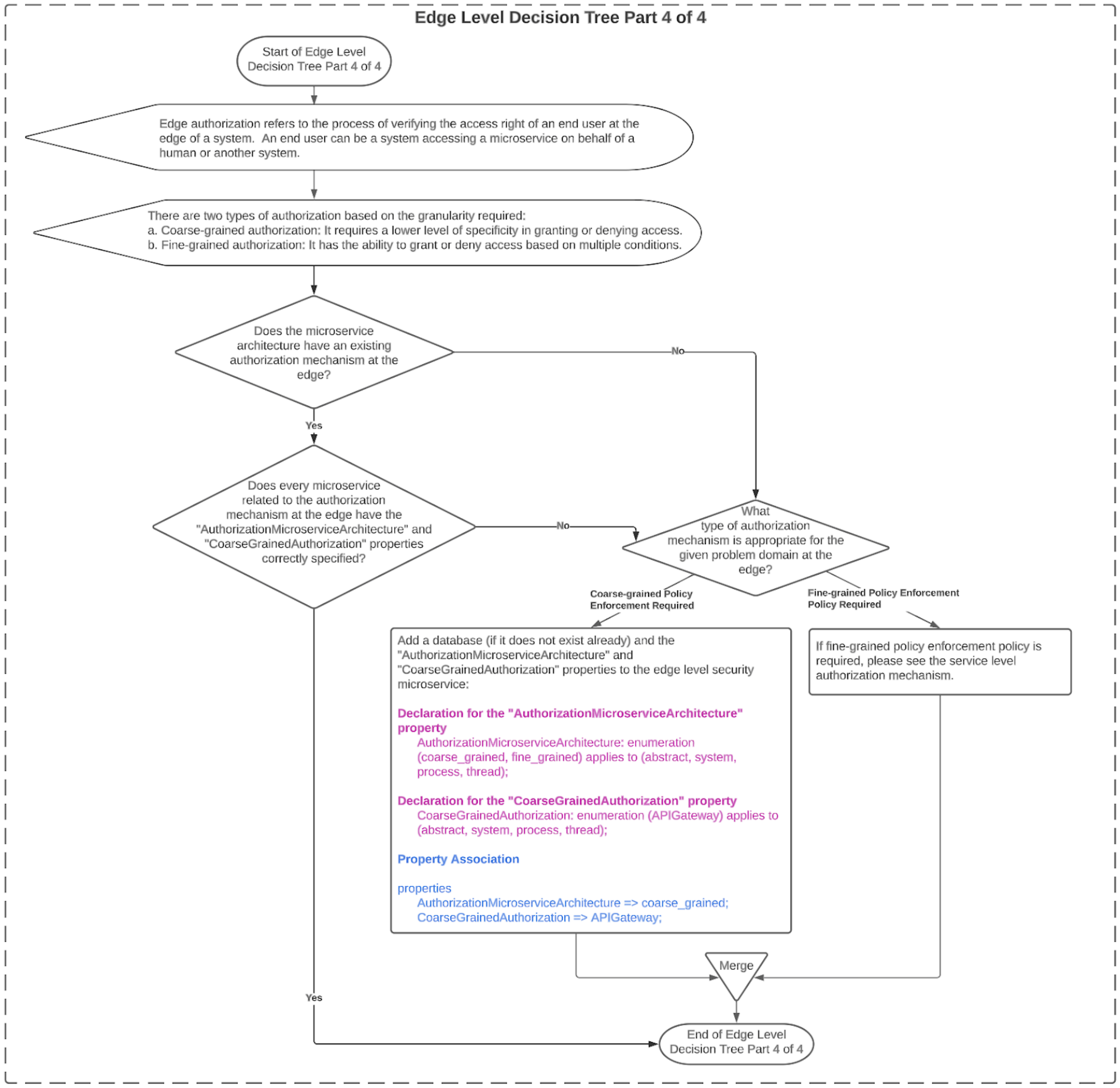
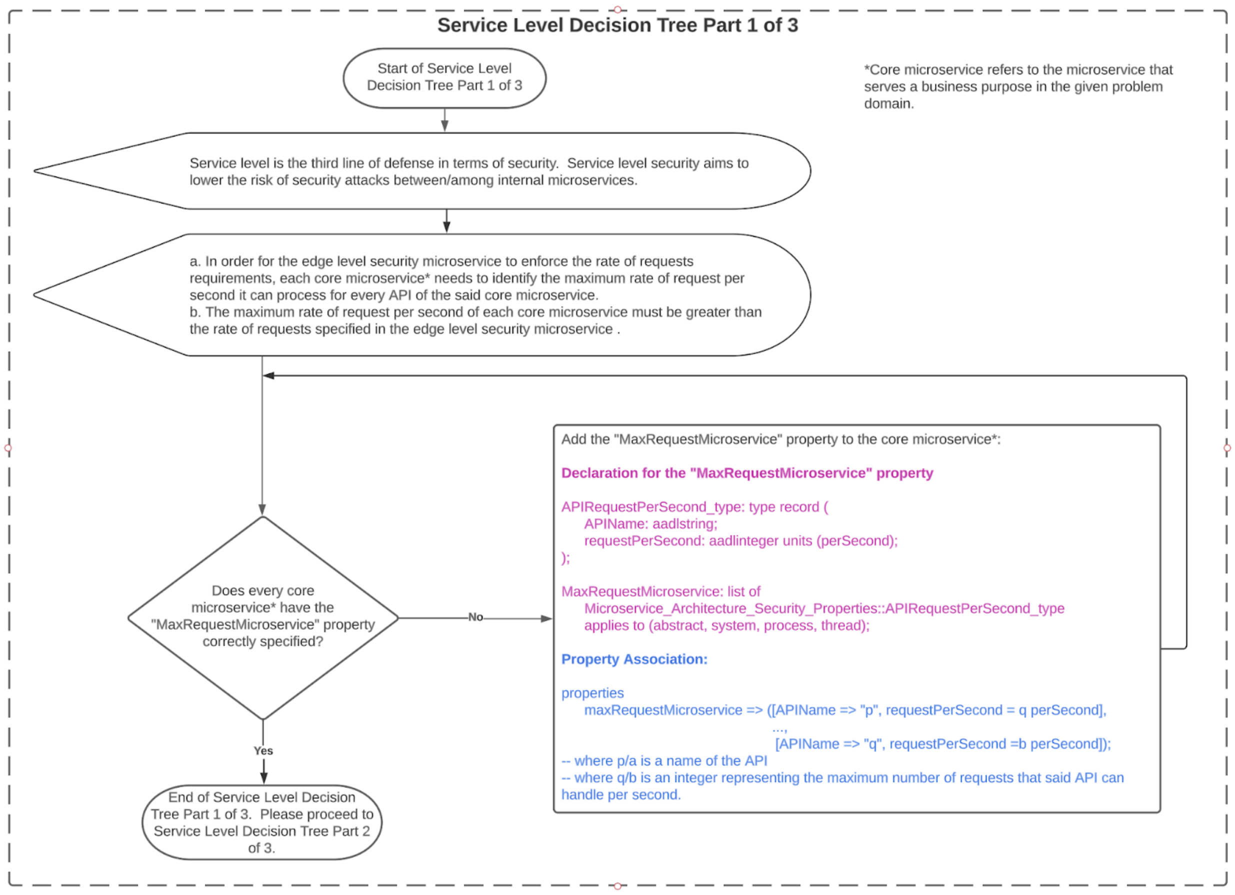

Edge Level Decision Tree Part 4 of 4
Start of Edge Level Decision Tree Part 4 of 4
Edge authorization refers to the process of verifying the access right of an end user at the edge of a system. An end user can be a system accessing a microservice on behalf of a human or another system.
There are two types of authorization based on the granularity required:
a. Coarse-grained authorization: It requires a lower level of specificity in granting or denying access.
b. Fine-grained authorization: It has the ability to grant or deny access based on multiple conditions.
Does the microservice architecture have an existing authorization mechanism at the edge?
No
Yes
Does every microservice related to the authorization mechanism at the edge have the "AuthorizationMicroserviceArchitecture" and "CoarseGrainedAuthorization" properties correctly specified?
No
What type of authorization mechanism is appropriate for the given problem domain at the edge?
Coarse-grained Policy Enforcement Required
Fine-grained Policy Enforcement Policy Required
Add a database (if it does not exist already) and the "AuthorizationMicroserviceArchitecture" and "CoarseGrainedAuthorization" properties to the edge level security microservice:
Declaration for the "AuthorizationMicroserviceArchitecture" property
AuthorizationMicroserviceArchitecture: enumeration (coarse_grained, fine_grained) applies to (abstract, system, process, thread);
Declaration for the "CoarseGrainedAuthorization" property
CoarseGrainedAuthorization: enumeration (APIGateway) applies to (abstract, system, process, thread);
Property Association
properties
AuthorizationMicroserviceArchitecture => coarse_grained;
CoarseGrainedAuthorization => APIGateway;
If fine-grained policy enforcement policy is required, please see the service level authorization mechanism.
Merge
Yes
End of Edge Level Decision Tree Part 4 of 4

Figure 19: Edge Level Perimeter Level Decision Tree Part 4 of 4

A.9. Service Level Perimeter Level Decision Tree Part 1 of 3

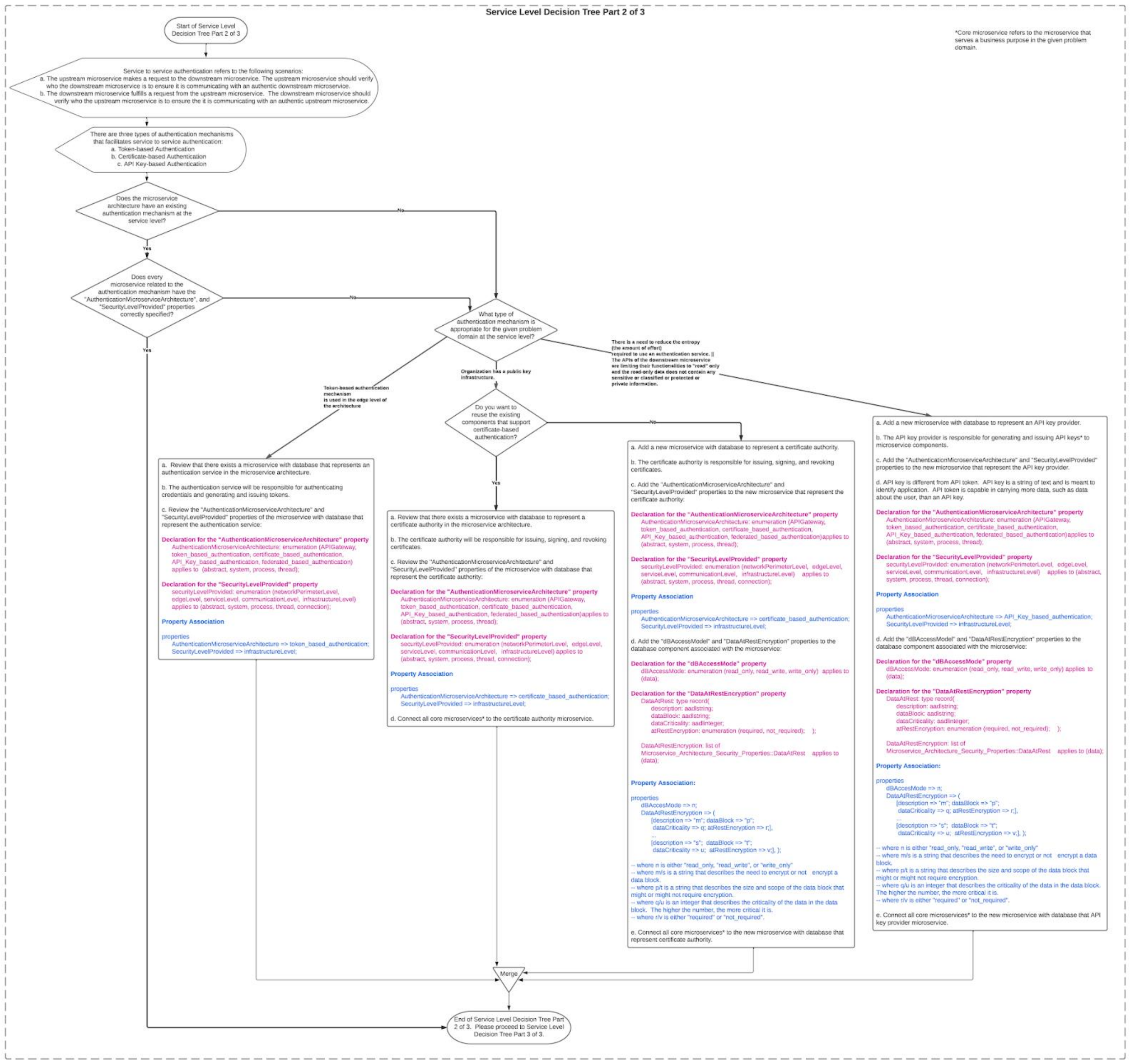

Service Level Decision Tree Part 1 of 3
Start of Service Level Decision Tree Part 1 of 3
*Core microservice refers to the microservice that serves a business purpose in the given problem domain.
Service level is the third line of defense in terms of security. Service level security aims to lower the risk of security attacks between/among internal microservices.
a. In order for the edge level security microservice to enforce the rate of requests requirements, each core microservice* needs to identify the maximum rate of request per second it can process for every API of the said core microservice.
b. The maximum rate of request per second of each core microservice must be greater than the rate of requests specified in the edge level security microservice .
Does every core microservice* have the "MaxRequestMicroservice" property correctly specified?
No
Yes
End of Service Level Decision Tree Part 1 of 3. Please proceed to Service Level Decision Tree Part 2 of 3.
Add the "MaxRequestMicroservice" property to the core microservice*:
Declaration for the "MaxRequestMicroservice" property
Property Association:

```
APIRequestPerSecond_type: type record (
    APIName: aadlstring;
    requestPerSecond: aadlinteger units (perSecond);
);

MaxRequestMicroservice: list of
    Microservice_Architecture_Security_Properties::APIRequestPerSecond_type
    applies to (abstract, system, process, thread);
```

```
properties
    maxRequestMicroservice => ([APIName => "p", requestPerSecond = q perSecond],
                                  ...,
                                  [APIName => "q", requestPerSecond =b perSecond]);
-- where p/a is a name of the API
-- where q/b is an integer representing the maximum number of requests that said API can
handle per second.
```

Figure 20: Service Level Perimeter Level Decision Tree Part 1 of 3

A.10. SERVICE LEVEL PERIMETER LEVEL DECISION TREE PART 2 OF 3


Service Level Decision Tree Part 2 of 3
*Core microservice refers to the microservice that serves a business purpose in the given problem domain.
Start of Service Level Decision Tree Part 2 of 3
Service to service authentication refers to the following scenarios:
a. The upstream microservice makes a request to the downstream microservice. The upstream microservice should verify who the downstream microservice is to ensure it is communicating with an authentic downstream microservice.
b. The downstream microservice fulfills a request from the upstream microservice. The downstream microservice should verify who the upstream microservice is to ensure the it is communicating with an authentic upstream microservice.
There are three types of authentication mechanisms that facilitates service to service authentication:
a. Token-based Authentication
b. Certificate-based Authentication
c. API Key-based Authentication
Does the microservice architecture have an existing authentication mechanism at the service level?
No
Yes
Does every microservice related to the authentication mechanism have the "AuthenticationMicroserviceArchitecture", and "SecurityLevelProvided" properties correctly specified?
No
Yes
What type of authentication mechanism is appropriate for the given problem domain at the service level?
Token-based authentication mechanism is used in the edge level of the architecture
Organization has a public key infrastructure.
There is a need to reduce the entropy (the amount of effort) required to use an authentication service. || The APIs of the downstream microservice are limiting their functionalities to "read" only and the read-only data does not contain any sensitive or classified or protected or private information.
Do you want to reuse the existing components that support certificate-based authentication?
No
Yes
Merge
End of Service Level Decision Tree Part 2 of 3. Please proceed to Service Level Decision Tree Part 3 of 3.

a. Review that there exists a microservice with database that represents an authentication service in the microservice architecture.

b. The authentication service will be responsible for authenticating credentials and generating and issuing tokens.

c. Review the "AuthenticationMicroserviceArchitecture" and "SecurityLevelProvided" properties of the microservice with database that represent the authentication service:

```
Declaration for the "AuthenticationMicroserviceArchitecture" property
    AuthenticationMicroserviceArchitecture: enumeration (APIGateway,
    token_based_authentication, certificate_based_authentication,
    API_Key_based_authentication, federated_based_authentication)
    applies to  (abstract, system, process, thread);

Declaration for the "SecurityLevelProvided" property
    securityLevelProvided: enumeration (networkPerimeterLevel,
    edgeLevel, serviceLevel, communicationLevel, infrastructureLevel)
    applies to (abstract, system, process, thread, connection);

Property Association

properties
    AuthenticationMicroserviceArchitecture => token_based_authentication;
    SecurityLevelProvided => infrastructureLevel;
```

a. Review that there exists a microservice with database to represent a certificate authority in the microservice architecture.

b. The certificate authority will be responsible for issuing, signing, and revoking certificates.

c. Review the "AuthenticationMicroserviceArchitecture" and "SecurityLevelProvided" properties of the microservice with database that represent the certificate authority:

```
Declaration for the "AuthenticationMicroserviceArchitecture" property
    AuthenticationMicroserviceArchitecture: enumeration (APIGateway,
    token_based_authentication, certificate_based_authentication,
    API_Key_based_authentication, federated_based_authentication)applies to
    (abstract, system, process, thread);

Declaration for the "SecurityLevelProvided" property
    securityLevelProvided: enumeration (networkPerimeterLevel,  edgeLevel,
    serviceLevel, communicationLevel,  infrastructureLevel) applies to
    (abstract, system, process, thread, connection);

Property Association

properties
    AuthenticationMicroserviceArchitecture => certificate_based_authentication;
    SecurityLevelProvided => infrastructureLevel;
```

d. Connect all core microservices* to the certificate authority microservice.

a. Add a new microservice with database to represent a certificate authority.

b. The certificate authority is responsible for issuing, signing, and revoking certificates.

c. Add the "AuthenticationMicroserviceArchitecture" and "SecurityLevelProvided" properties to the new microservice that represent the certificate authority:

```
Declaration for the "AuthenticationMicroserviceArchitecture" property
    AuthenticationMicroserviceArchitecture: enumeration (APIGateway,
    token_based_authentication, certificate_based_authentication,
    API_Key_based_authentication, federated_based_authentication)applies to
    (abstract, system, process, thread);

Declaration for the "SecurityLevelProvided" property
    securityLevelProvided: enumeration (networkPerimeterLevel,  edgeLevel,
    serviceLevel, communicationLevel,   infrastructureLevel)    applies to
    (abstract, system, process, thread, connection);

Property Association

properties
    AuthenticationMicroserviceArchitecture => certificate_based_authentication;
    SecurityLevelProvided => infrastructureLevel;
```

d. Add the "dBAccessModel" and "DataAtRestEncryption" properties to the database component associated with the microservice:

```
Declaration for the "dBAccessMode" property
    dBAccessMode: enumeration (read_only, read_write, write_only)  applies to
    (data);

Declaration for the "DataAtRestEncryption" property
    DataAtRest: type record(
        description: aadlstring;
        dataBlock: aadlstring;
        dataCriticality: aadlinteger;
        atRestEncryption: enumeration (required, not_required);    );

    DataAtRestEncryption: list of
    Microservice_Architecture_Security_Properties::DataAtRest    applies to
    (data);


Property Association:

properties
    dBAccesMode => n;
    DataAtRestEncryption => (
        [description => "m"; dataBlock => "p";
         dataCriticality => q; atRestEncryption => r;],
        ...
        [description => "s";  dataBlock => "t";
         dataCriticality => u;  atRestEncryption => v;], );

-- where n is either "read_only, "read_write", or "write_only"
-- where m/s is a string that describes the need to encrypt or not   encrypt a
data block.
-- where p/t is a string that describes the size and scope of the data block that
might or might not require encryption.
-- where q/u is an integer that describes the criticality of the data in the data
block.  The higher the number, the more critical it is.
-- where r/v is either "required" or "not_required".
```

e. Connect all core microservices* to the new microservice with database that represent certificate authority.

a. Add a new microservice with database to represent an API key provider.

b. The API key provider is responsible for generating and issuing API keys* to microservice components.

c. Add the "AuthenticationMicroserviceArchitecture" and "SecurityLevelProvided" properties to the new microservice that represent the API key provider.

d. API key is different from API token. API key is a string of text and is meant to identify application. API token is capable in carrying more data, such as data about the user, than an API key.

```
Declaration for the "AuthenticationMicroserviceArchitecture" property
    AuthenticationMicroserviceArchitecture: enumeration (APIGateway,
    token_based_authentication, certificate_based_authentication,
    API_Key_based_authentication, federated_based_authentication)applies to
    (abstract, system, process, thread);

Declaration for the "SecurityLevelProvided" property
    securityLevelProvided: enumeration (networkPerimeterLevel,  edgeLevel,
    serviceLevel, communicationLevel,   infrastructureLevel)    applies to (abstract,
    system, process, thread, connection);

Property Association

properties
    AuthenticationMicroserviceArchitecture => API_Key_based_authentication;
    SecurityLevelProvided => infrastructureLevel;
```

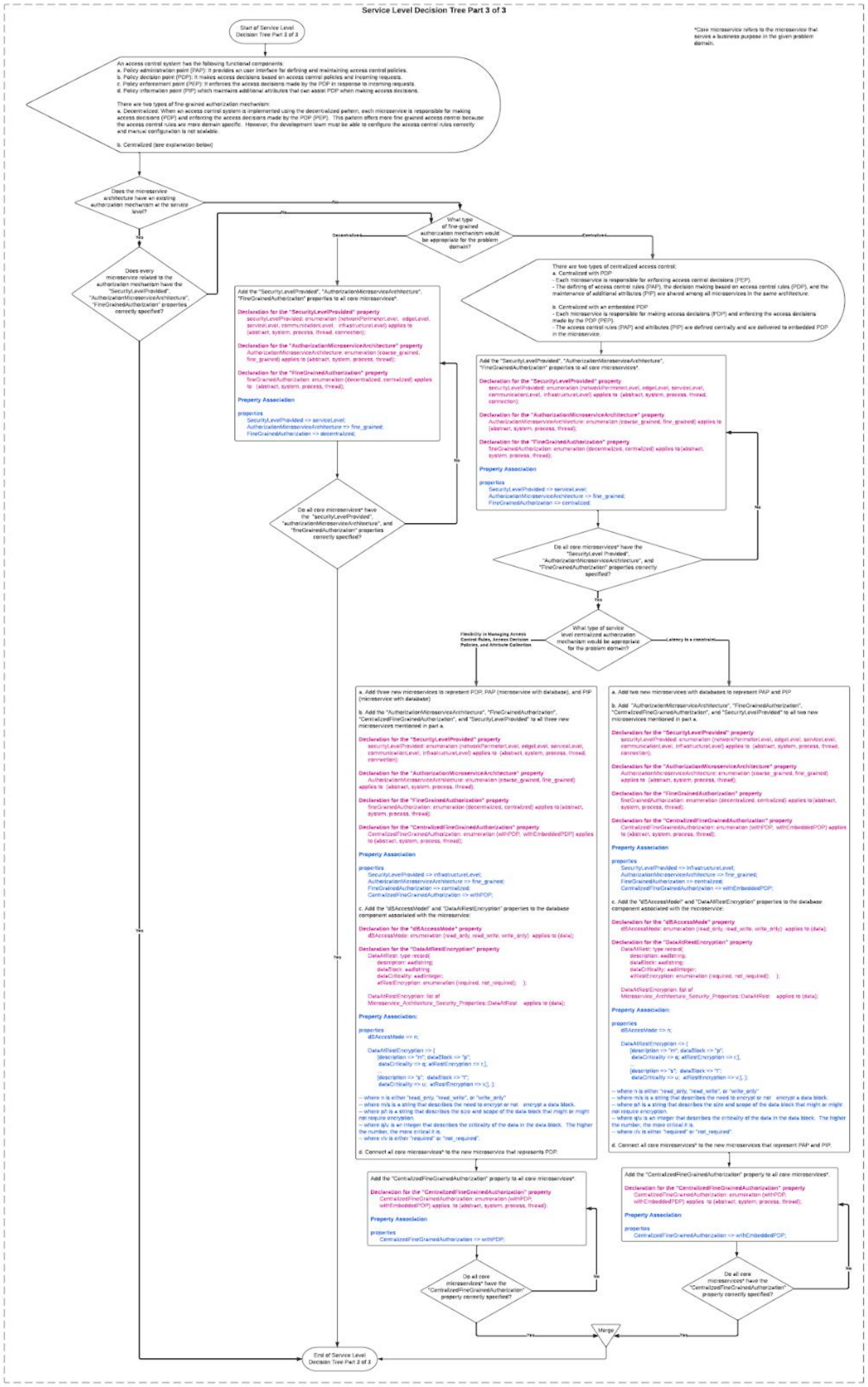
d. Add the "dBAccessModel" and "DataAtRestEncryption" properties to the database component associated with the microservice:

```
Declaration for the "dBAccessMode" property
    dBAccessMode: enumeration (read_only, read_write, write_only) applies to
    (data);

Declaration for the "DataAtRestEncryption" property
    DataAtRest: type record(
        description: aadlstring;
        dataBlock: aadlstring;
        dataCriticality: aadlinteger;
        atRestEncryption: enumeration (required, not_required);    );

    DataAtRestEncryption: list of
    Microservice_Architecture_Security_Properties::DataAtRest    applies to (data);

Property Association:

properties
    dBAccesMode => n;
    DataAtRestEncryption => (
        [description => "m"; dataBlock => "p";
         dataCriticality => q; atRestEncryption => r;],
        ...
        [description => "s";  dataBlock => "t";
         dataCriticality => u;  atRestEncryption => v;], );

-- where n is either "read_only, "read_write", or "write_only"
-- where m/s is a string that describes the need to encrypt or not   encrypt a data
block.
-- where p/t is a string that describes the size and scope of the data block that
might or might not require encryption.
-- where q/u is an integer that describes the criticality of the data in the data block.
The higher the number, the more critical it is.
-- where r/v is either "required" or "not_required".
```

e. Connect all core microservices* to the new microservice with database that API key provider microservice.

Figure 21: Service Level Perimeter Level Decision Tree Part 2 of 3

A.11. Service Level Perimeter Level Decision Tree Part 3 of 3



Service Level Decision Tree Part 3 of 3
Start of Service Level Decision Tree Part 3 of 3
*Core microservice refers to the microservice that serves a business purpose in the given problem domain.
Does the microservice architecture have an existing authorization mechanism at the service level?
No
Yes
Does every microservice related to the authorization mechanism have the "SecurityLevelProvided", "AuthorizationMicroserviceArchitecture", "FineGrainedAuthorization" properties correctly specified?
No
What type of fine-grained authorization mechanism would be appropriate for the problem domain?
Decentralized
Centralized
Do all core microservices* have the "securityLevelProvided", "authorizationMicroserviceArchitecture", and "fineGrainedAuthorization" properties correctly specified?
No
Do all core microservices* have the "SecurityLevel Provided", "AuthorizationMicroserviceArchitecture", and "FineGrainedAuthorization" properties correctly specified?
No
Yes
What type of service level centralized authorization mechanism would be appropriate for the problem domain?
Flexibility in Managing Access Control Rules, Access Decision Policies, and Attribute Collection
Latency is a constraint
Do all core microservices* have the "CentralizedFineGrainedAuthorization" property correctly specified?
No
Yes
Do all core microservices* have the "CentralizedFineGrainedAuthorization" property correctly specified?
No
Yes
Merge
Yes
Yes
End of Service Level Decision Tree Part 3 of 3

An access control system has the following functional components:
a. Policy administration point (PAP): it provides an user interface for defining and maintaining access control policies.
b. Policy decision point (PDP): It makes access decisions based on access control policies and incoming requests.
c. Policy enforcement point (PEP): It enforces the access decisions made by the PDP in response to incoming requests.
d. Policy information point (PIP) which maintains additional attributes that can assist PDP when making access decisions.

There are two types of fine-grained authorization mechanism:
a. Decentralized: When an access control system is implemented using the decentralized pattern, each microservice is responsible for making access decisions (PDP) and enforcing the access decisions made by the PDP (PEP). This pattern offers more fine grained access control because the access control rules are more domain specific. However, the development team must be able to configure the access control rules correctly and manual configuration is not scalable.

b. Centralized (see explanation below)

Add the "SecurityLevelProvided", "AuthorizationMicroserviceArchitecture", "FineGrainedAuthorization" properties to all core microservices*.

```
Declaration for the "SecurityLevelProvided" property
    securityLevelProvided: enumeration (networkPerimeterLevel,  edgeLevel,
    serviceLevel, communicationLevel,  infrastructureLevel) applies to
    (abstract, system, process, thread, connection);

Declaration for the "AuthorizationMicroserviceArchitecture" property
    AuthorizationMicroserviceArchitecture: enumeration (coarse_grained,
    fine_grained) applies to (abstract, system, process, thread);

Declaration for the "FineGrainedAuthorization" property
    fineGrainedAuthorization: enumeration (decentralized, centralized) applies
    to  (abstract, system, process, thread);

Property Association

properties
    SecurityLevelProvided => serviceLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => decentralized;
```

There are two types of centralized access control:
a. Centralized with PDP
- Each microservice is responsible for enforcing access control decisions (PEP).
- The defining of access control rules (PAP), the decision making based on access control rules (PDP), and the maintenance of additional attributes (PIP) are shared among all microservices in the same architecture.

b. Centralized with an embedded PDP
- Each microservice is responsible for making access decisions (PDP) and enforcing the access decisions made by the PDP (PEP).
- The access control rules (PAP) and attributes (PIP) are defined centrally and are delivered to embedded PDP in the microservice.

Add the "SecurityLevelProvided", "AuthorizationMicroserviceArchitecture", "FineGrainedAuthorization" properties to all core microservices*.

```
Declaration for the "SecurityLevelProvided" property
    securityLevelProvided: enumeration (networkPerimeterLevel, edgeLevel, serviceLevel,
    communicationLevel, infrastructureLevel) applies to (abstract, system, process, thread,
    connection);

Declaration for the "AuthorizationMicroserviceArchitecture" property
    AuthorizationMicroserviceArchitecture: enumeration (coarse_grained, fine_grained) applies to
    (abstract, system, process, thread);

Declaration for the "FineGrainedAuthorization" property
    fineGrainedAuthorization: enumeration (decentralized, centralized) applies to (abstract,
    system, process, thread);

Property Association

properties
    SecurityLevelProvided => serviceLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => centralized;
```

a. Add three new microservices to represent PDP, PAP (microservice with database), and PIP (microservice with database)

b. Add the "AuthorizationMicroserviceArchitecture", "FineGrainedAuthorization", "CentralizedFineGrainedAuthorization", and "SecurityLevelProvided" to all three new microservices mentioned in part a.

```
Declaration for the "SecurityLevelProvided" property
    securityLevelProvided: enumeration (networkPerimeterLevel, edgeLevel, serviceLevel,
    communicationLevel, infrastructureLevel) applies to (abstract, system, process, thread,
    connection);

Declaration for the "AuthorizationMicroserviceArchitecture" property
    AuthorizationMicroserviceArchitecture: enumeration (coarse_grained, fine_grained)
applies to (abstract, system, process, thread);

Declaration for the "FineGrainedAuthorization" property
    fineGrainedAuthorization: enumeration (decentralized, centralized) applies to (abstract,
    system, process, thread);

Declaration for the "CentralizedFineGrainedAuthorization" property
    CentralizedFineGrainedAuthorization: enumeration (withPDP, withEmbeddedPDP) applies
    to (abstract, system, process, thread);

Property Association

properties
    SecurityLevelProvided => infrastructureLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => centralized;
    CentralizedFineGrainedAuthorization => withPDP;
```

c. Add the "dBAccessModel" and "DataAtRestEncryption" properties to the database component associated with the microservice:

```
Declaration for the "dBAccessMode" property
    dBAccessMode: enumeration (read_only, read_write, write_only) applies to (data);

Declaration for the "DataAtRestEncryption" property
    DataAtRest: type record(
        description: aadlstring;
        dataBlock: aadlstring;
        dataCriticality: aadlinteger;
        atRestEncryption: enumeration (required, not_required);    );

    DataAtRestEncryption: list of
    Microservice_Architecture_Security_Properties::DataAtRest    applies to (data);

Property Association:

properties
    dBAccessMode => n;

    DataAtRestEncryption => (
        [description => "m"; dataBlock => "p";
         dataCriticality => q; atRestEncryption => r;],
        ...
        [description => "s"; dataBlock => "t";
         dataCriticality => u; atRestEncryption => v;], );

-- where n is either "read_only, "read_write", or "write_only"
-- where m/s is a string that describes the need to encrypt or not   encrypt a data block.
-- where p/t is a string that describes the size and scope of the data block that might or might
not require encryption.
-- where q/u is an integer that describes the criticality of the data in the data block.  The higher
the number, the more critical it is.
-- where r/v is either "required" or "not_required".
```

d. Connect all core microservices* to the new microservice that represents PDP.

a. Add two new microservices with databases to represent PAP and PIP

b. Add "AuthorizationMicroserviceArchitecture", "FineGrainedAuthorization", "CentralizedFineGrainedAuthorization", and "SecurityLevelProvided" to all two new microservices mentioned in part a.

```
Declaration for the "SecurityLevelProvided" property
    securityLevelProvided: enumeration (networkPerimeterLevel, edgeLevel, serviceLevel,
    communicationLevel, infrastructureLevel) applies to (abstract, system, process, thread,
    connection);

Declaration for the "AuthorizationMicroserviceArchitecture" property
    AuthorizationMicroserviceArchitecture: enumeration (coarse_grained, fine_grained)
    applies to (abstract, system, process, thread);

Declaration for the "FineGrainedAuthorization" property
    fineGrainedAuthorization: enumeration (decentralized, centralized) applies to (abstract,
    system, process, thread);

Declaration for the "CentralizedFineGrainedAuthorization" property
    CentralizedFineGrainedAuthorization: enumeration (withPDP, withEmbeddedPDP) applies
    to (abstract, system, process, thread);

Property Association

properties
    SecurityLevelProvided => infrastructureLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => centralized;
    CentralizedFineGrainedAuthorization => withEmbeddedPDP;
```

c. Add the "dBAccessModel" and "DataAtRestEncryption" properties to the database component associated with the microservice:

```
Declaration for the "dBAccessMode" property
    dBAccessMode: enumeration (read_only, read_write, write_only) applies to (data);

Declaration for the "DataAtRestEncryption" property
    DataAtRest: type record(
        description: aadlstring;
        dataBlock: aadlstring;
        dataCriticality: aadlinteger;
        atRestEncryption: enumeration (required, not_required);    );

    DataAtRestEncryption: list of
    Microservice_Architecture_Security_Properties::DataAtRest    applies to (data);

Property Association:

properties
    dBAccessMode => n;

    DataAtRestEncryption => (
        [description => "m"; dataBlock => "p";
         dataCriticality => q; atRestEncryption => r;],
        ...
        [description => "s"; dataBlock => "t";
         dataCriticality => u; atRestEncryption => v;], );

-- where n is either "read_only, "read_write", or "write_only"
-- where m/s is a string that describes the need to encrypt or not   encrypt a data block.
-- where p/t is a string that describes the size and scope of the data block that might or might
not require encryption.
-- where q/u is an integer that describes the criticality of the data in the data block.  The higher
the number, the more critical it is.
-- where r/v is either "required" or "not_required".
```

d. Connect all core microservices* to the new microservices that represent PAP and PIP.

Add the "CentralizedFineGrainedAuthorization" property to all core microservices*.

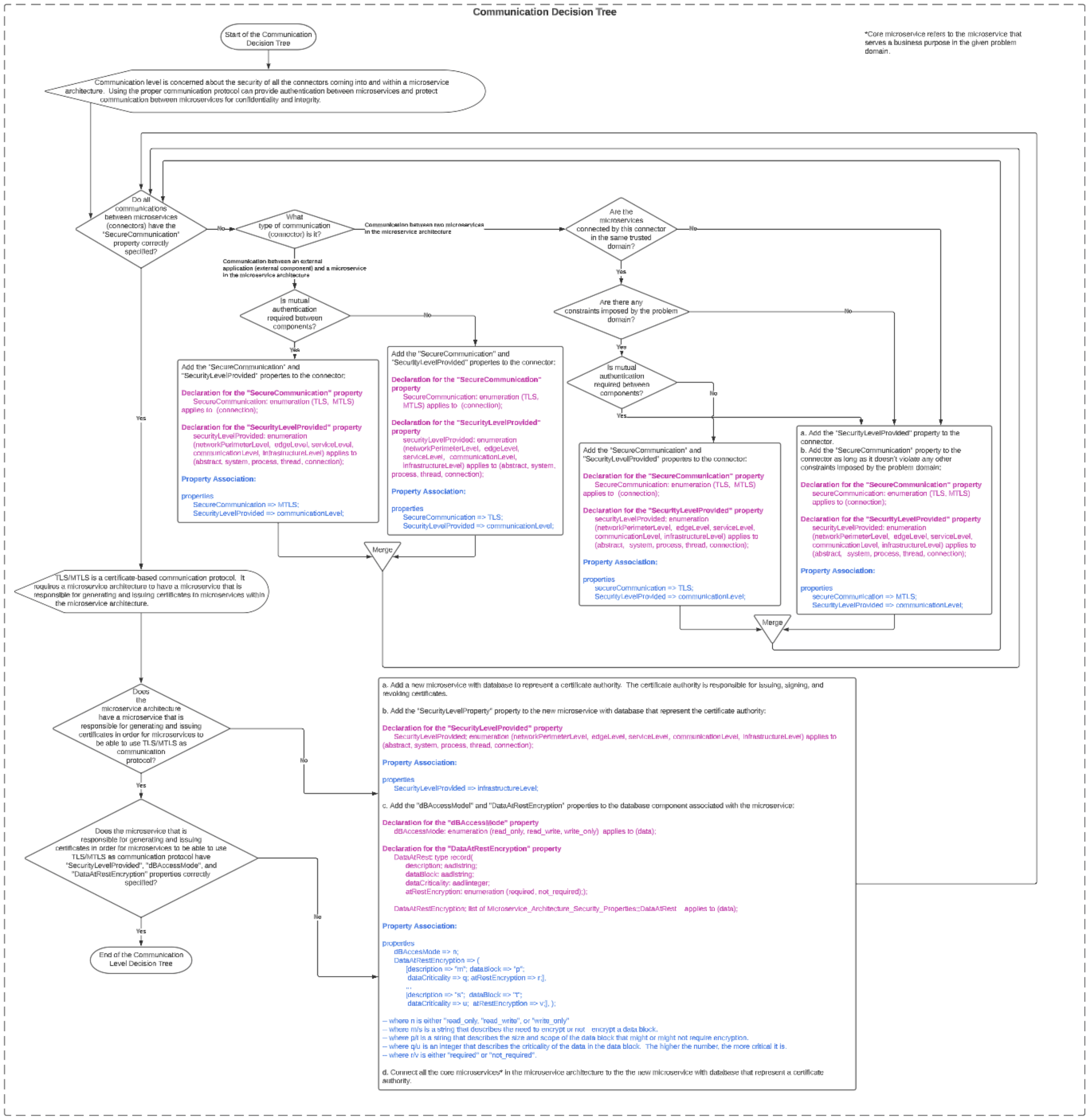
```
Declaration for the "CentralizedFineGrainedAuthorization" property
    CentralizedFineGrainedAuthorization: enumeration (withPDP,
    withEmbeddedPDP) applies to (abstract, system, process, thread);

Property Association

properties
    CentralizedFineGrainedAuthorization => withPDP;
```

Add the "CentralizedFineGrainedAuthorization" property to all core microservices*.

```
Declaration for the "CentralizedFineGrainedAuthorization" property
    CentralizedFineGrainedAuthorization: enumeration (withPDP,
    withEmbeddedPDP) applies to (abstract, system, process, thread);

Property Association

properties
    CentralizedFineGrainedAuthorization => withEmbeddedPDP;
```

Figure 22: Service Level Perimeter Level Decision Tree Part 3 of 3

A.12. Communication Decision Tree



Communication Decision Tree
*Core microservice refers to the microservice that serves a business purpose in the given problem domain.
Start of the Communication Decision Tree
Communication level is concerned about the security of all the connectors coming into and within a microservice architecture. Using the proper communication protocol can provide authentication between microservices and protect communication between microservices for confidentiality and integrity.
Do all communications between microservices (connectors) have the "SecureCommunication" property correctly specified?
No
Yes
What type of communication (connector) is it?
Communication between two microservices in the microservice architecture
Communication between an external application (external component) and a microservice in the microservice architecture
Is mutual authentication required between components?
No
Yes
Are the microservices connected by this connector in the same trusted domain?
No
Yes
Are there any constraints imposed by the problem domain?
No
Yes
Is mutual authentication required between components?
No
Yes
Merge
Merge
TLS/MTLS is a certificate-based communication protocol. It requires a microservice architecture to have a microservice that is responsible for generating and issuing certificates to microservices within the microservice architecture.
Does the microservice architecture have a microservice that is responsible for generating and issuing certificates in order for microservices to be able to use TLS/MTLS as communication protocol?
No
Yes
Does the microservice that is responsible for generating and issuing certificates in order for microservices to be able to use TLS/MTLS as communication protocol have "SecurityLevelProvided", "dBAccessMode", and "DataAtRestEncryption" properties correctly specified?
No
Yes
End of the Communication Level Decision Tree

Add the "SecureCommunication" and "SecurityLevelProvided" propertes to the connector;

Declaration for the "SecureCommunication" property

```
    SecureCommunication: enumeration (TLS,  MTLS)
applies to  (connection);
```

Declaration for the "SecurityLevelProvided" property

```
    securityLevelProvided: enumeration
    (networkPerimeterLevel,  edgeLevel, serviceLevel,
    communicationLevel, infrastructureLevel) applies to
    (abstract, system, process, thread, connection);
```

Property Association:

```
properties
    SecureCommunication => MTLS;
    SecurityLevelProvided => communicationLevel;
```

Add the "SecureCommunication" and "SecurityLevelProvided" propertes to the connector:

Declaration for the "SecureCommunication" property

```
    SecureCommunication: enumeration (TLS,
    MTLS) applies to  (connection);
```

Declaration for the "SecurityLevelProvided" property

```
    securityLevelProvided: enumeration
    (networkPerimeterLevel,  edgeLevel,
    serviceLevel,  communicationLevel,
    infrastructureLevel) applies to (abstract, system,
process, thread, connection);
```

Property Association:

```
properties
    SecureCommunication => TLS;
    SecurityLevelProvided => communicationLevel;
```

Add the "SecureCommunication" and "SecurityLevelProvided" propertes to the connector:

Declaration for the "SecureCommunication" property

```
    SecureCommunication: enumeration (TLS,  MTLS)
applies to  (connection);
```

Declaration for the "SecurityLevelProvided" property

```
    securityLevelProvided; enumeration
    (networkPerimeterLevel,  edgeLevel, serviceLevel,
    communicationLevel, infrastructureLevel) applies to
    (abstract,   system, process, thread, connection);
```

Property Association:

```
properties
    secureCommunication => TLS;
    SecurityLevelProvided => communicationLevel;
```

a. Add the "SecurityLevelProvided" property to the connector.
b. Add the "SecureCommunication" property to the connector as long as it doesn't violate any other constraints imposed by the problem domain:

Declaration for the "SecureCommunication" property

```
    secureCommunication: enumeration (TLS, MTLS)
    applies to (connection);
```

Declaration for the "SecurityLevelProvided" property

```
    securityLevelProvided: enumeration
    (networkPerimeterLevel,  edgeLevel, serviceLevel,
    communicationLevel, infrastructureLevel) applies to
    (abstract,   system, process, thread, connection);
```

Property Association:

```
properties
    secureCommunication => MTLS;
    SecurityLevelProvided => communicationLevel;
```

a. Add a new microservice with database to represent a certificate authority. The certificate authority is responsible for issuing, signing, and revoking certificates.

b. Add the "SecurityLevelProperty" property to the new microservice with database that represent the certificate authority:

Declaration for the "SecurityLevelProvided" property

```
    SecurityLevelProvided; enumeration (networkPerimeterLevel,  edgeLevel, serviceLevel,  communicationLevel,  infrastructureLevel) applies to
(abstract, system, process, thread, connection);
```

Property Association:

```
properties
    SecurityLevelProvided => infrastructureLevel;
```

c. Add the "dBAccessModel" and "DataAtRestEncryption" properties to the database component associated with the microservice:

Declaration for the "dBAccessMode" property

```
    dBAccessMode: enumeration (read_only, read_write, write_only)  applies to (data);
```

Declaration for the "DataAtRestEncryption" property

```
    DataAtRest: type record(
        description; aadlstring;
        dataBlock: aadlstring;
        dataCriticality: aadlinteger;
        atRestEncryption: enumeration (required , not_required););

    DataAtRestEncryption; list of Microservice_Architecture_Security_Properties;;DataAtRest    applies to (data);
```

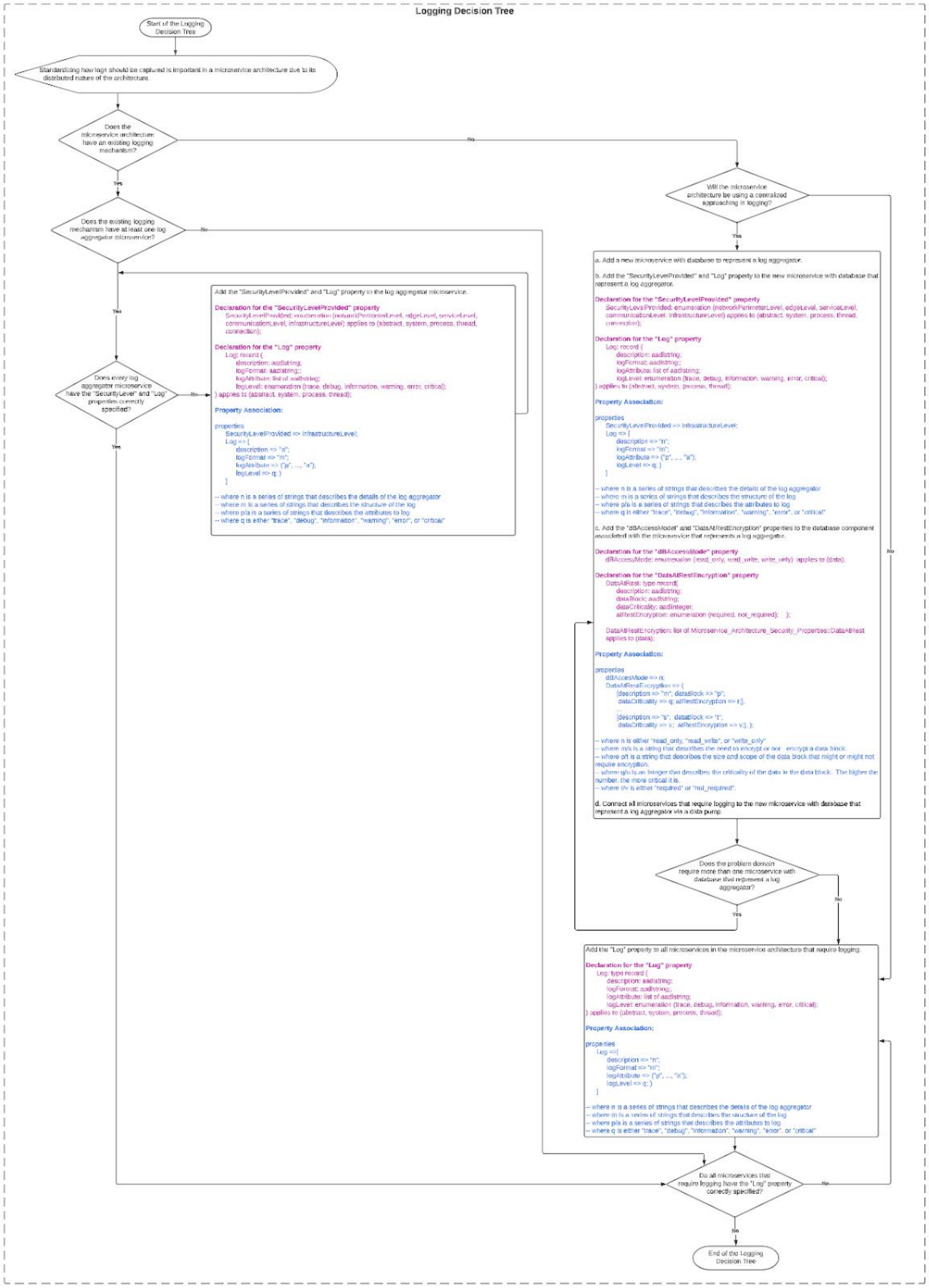
Property Association:

```
properties
    dBAccesMode => n;
    DataAtRestEncryption => (
        [description => "m"; dataBlock => "p";
         dataCriticality => q; atRestEncryption => r;],
        ...
        [description => "s";  dataBlock => "t";
         dataCriticality => u;  atRestEncryption => v;], );

-- where n is either "read_only, "read_write", or "write_only"
-- where m/s is a string that describes the need to encrypt or not    encrypt a data block.
-- where p/t is a string that describes the size and scope of the data block that might or might not require encryption.
-- where q/u is an integer that describes the criticality of the data in the data block.  The higher the number, the more critical it is.
-- where r/v is either "required" or "not_required".
```

d. Connect all the core microservices* in the microservice architecture to the the new microservice with database that represent a certificate authority.

Figure 23: Communication Decision Tree

A.13. Logging Decision Tree


Logging Decision Tree
Start of the Logging Decision Tree
Standardizing how logs should be captured is important in a microservice architecture due to its distributed nature of the architecture.
Does the microservice architecture have an existing logging mechanism?
No
Yes
Will the microservice architecture be using a centralized approaching in logging?
Yes
No
Does the existing logging mechanism have at least one log aggregator microservice?
No
Yes
Does every log aggregator microservice have the "SecurityLevel" and "Log" properties correctly specified?
No
Yes
Does the problem domain require more than one microservice with database that represent a log aggregator?
Yes
No
Do all microservices that require logging have the "Log" property correctly specified?
No
No
End of the Logging Decision Tree

Add the "SecurityLevelProvided" and "Log" property to the log aggregator microservice.

```
Declaration for the "SecurityLevelProvided" property
    SecurityLevelProvided: enumeration (networkPerimeterLevel, edgeLevel, serviceLevel,
    communicationLevel, infrastructureLevel) applies to (abstract, system, process, thread,
    connection);

Declaration for the "Log" property
    Log: record (
        description: aadlstring;
        logFormat: aadlstring;;
        logAttribute: list of aadlstring;
        logLevel: enumeration (trace, debug, information, warning, error, critical);
) applies to (abstract, system, process, thread);

Property Association:

properties
    SecurityLevelProvided => infrastructureLevel;
    Log => [
        description => "n";
        logFormat => "m";
        logAttribute => ("p", ..., "a");
        logLevel => q; )
    ]

-- where n is a series of strings that describes the details of the log aggregator
-- where m is a series of strings that describes the structure of the log
-- where p/a is a series of strings that describes the attributes to log
-- where q is either "trace", "debug", "information", "warning", "error", or "critical"
```

a. Add a new microservice with database to represent a log aggregator.

b. Add the "SecurityLevelProvided" and "Log" property to the new microservice with database that represent a log aggregator.

```
Declaration for the "SecurityLevelProvided" property
    SecurityLevelProvided: enumeration (networkPerimeterLevel, edgeLevel, serviceLevel,
    communicationLevel, infrastructureLevel) applies to (abstract, system, process, thread,
    connection);

Declaration for the "Log" property
    Log: record (
        description: aadlstring;
        logFormat: aadlstring;;
        logAttribute: list of aadlstring;
        logLevel: enumeration (trace, debug, information, warning, error, critical);
) applies to (abstract, system, process, thread);

Property Association:

properties
    SecurityLevelProvided => infrastructureLevel;
    Log => [
        description => "n";
        logFormat => "m";
        logAttribute => ("p", ..., "a");
        logLevel => q; )
    ]

-- where n is a series of strings that describes the details of the log aggregator
-- where m is a series of strings that describes the structure of the log
-- where p/a is a series of strings that describes the attributes to log
-- where q is either "trace", "debug", "information", "warning", "error", or "critical"
```

c. Add the "dBAccessModel" and "DataAtRestEncryption" properties to the database component associated with the microservice that represents a log aggregator.

```
Declaration for the "dBAccessMode" property
    dBAccessMode: enumeration (read_only, read_write, write_only) applies to (data);

Declaration for the "DataAtRestEncryption" property
    DataAtRest: type record(
        description: aadlstring;
        dataBlock: aadlstring;
        dataCriticality: aadlinteger;
        atRestEncryption: enumeration (required, not_required);    );

    DataAtRestEncryption: list of Microservice_Architecture_Security_Properties::DataAtRest
    applies to (data);

Property Association:

properties
    dBAccesMode => n;
    DataAtRestEncryption => (
        [description => "m"; dataBlock => "p";
         dataCriticality => q; atRestEncryption => r],
        ...
        [description => "s";  dataBlock => "t";
         dataCriticality => u;  atRestEncryption => v], );

-- where n is either "read_only, "read_write", or "write_only"
-- where m/s is a string that describes the need to encrypt or not   encrypt a data block.
-- where p/t is a string that describes the size and scope of the data block that might or might not
require encryption.
-- where q/u is an integer that describes the criticality of the data in the data block.  The higher the
number, the more critical it is.
-- where r/v is either "required" or "not_required".
```

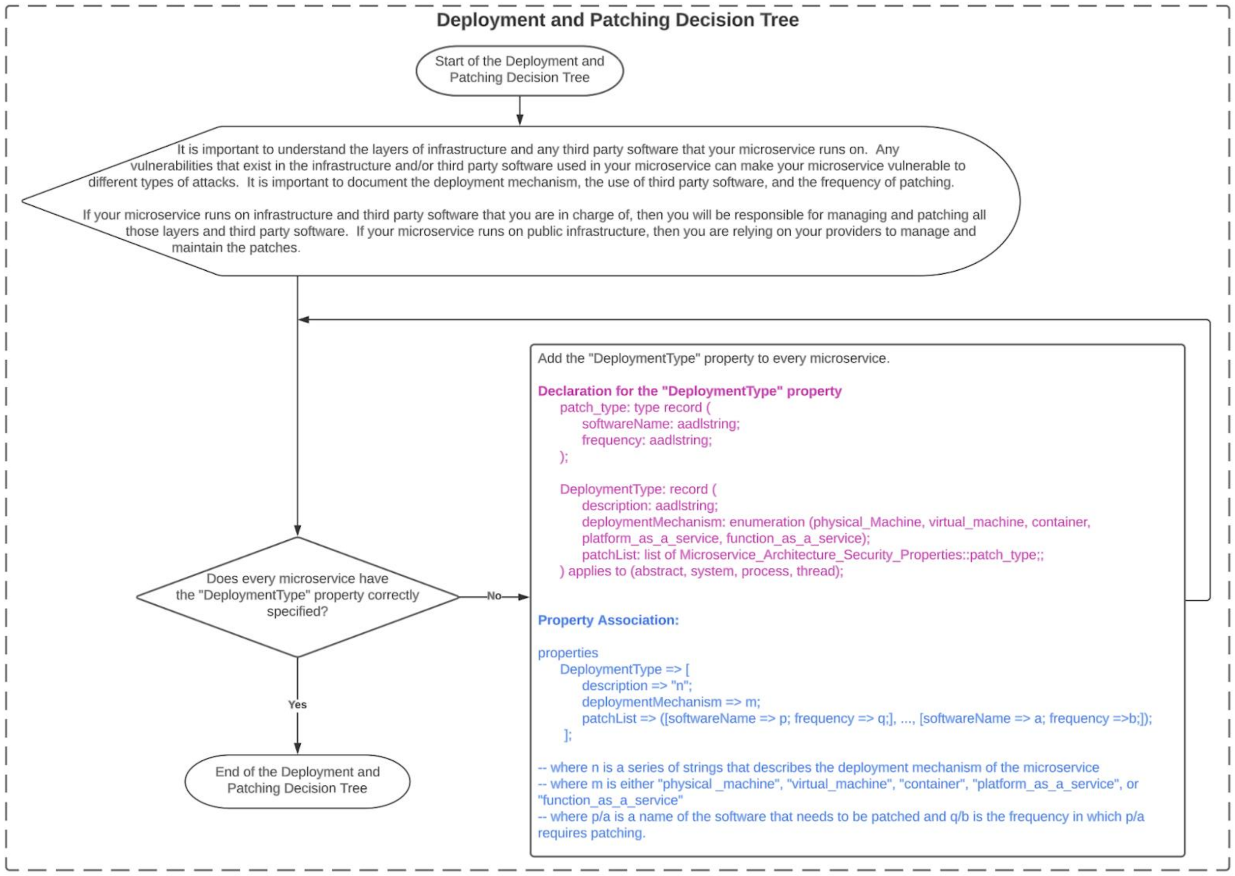
d. Connect all microservices that require logging to the new microservice with database that represent a log aggregator via a data pump.

Add the "Log" property to all microservices in the microservice architecture that require logging.

```
Declaration for the "Log" property
    Log: type record (
        description: aadlstring;
        logFormat: aadlstring;;
        logAttribute: list of aadlstring;
        logLevel: enumeration (trace, debug, information, warning, error, critical);
) applies to (abstract, system, process, thread);

Property Association:

properties
    Log =>[
        description => "n";
        logFormat => "m";
        logAttribute => ("p", ..., "a");
        logLevel => q; )
    ]

-- where n is a series of strings that describes the details of the log aggregator
-- where m is a series of strings that describes the structure of the log
-- where p/a is a series of strings that describes the attributes to log
-- where q is either "trace", "debug", "information", "warning", "error", or "critical"
```

Figure 24: Logging Decision Tree

A.14. Deployment and Patching Decision Tree



Deployment and Patching Decision Tree
Start of the Deployment and Patching Decision Tree
Does every microservice have the "DeploymentType" property correctly specified?
No
Yes
End of the Deployment and Patching Decision Tree

It is important to understand the layers of infrastructure and any third party software that your microservice runs on. Any vulnerabilities that exist in the infrastructure and/or third party software used in your microservice can make your microservice vulnerable to different types of attacks. It is important to document the deployment mechanism, the use of third party software, and the frequency of patching.

If your microservice runs on infrastructure and third party software that you are in charge of, then you will be responsible for managing and patching all those layers and third party software. If your microservice runs on public infrastructure, then you are relying on your providers to manage and maintain the patches.

Add the "DeploymentType" property to every microservice.

Declaration for the "DeploymentType" property

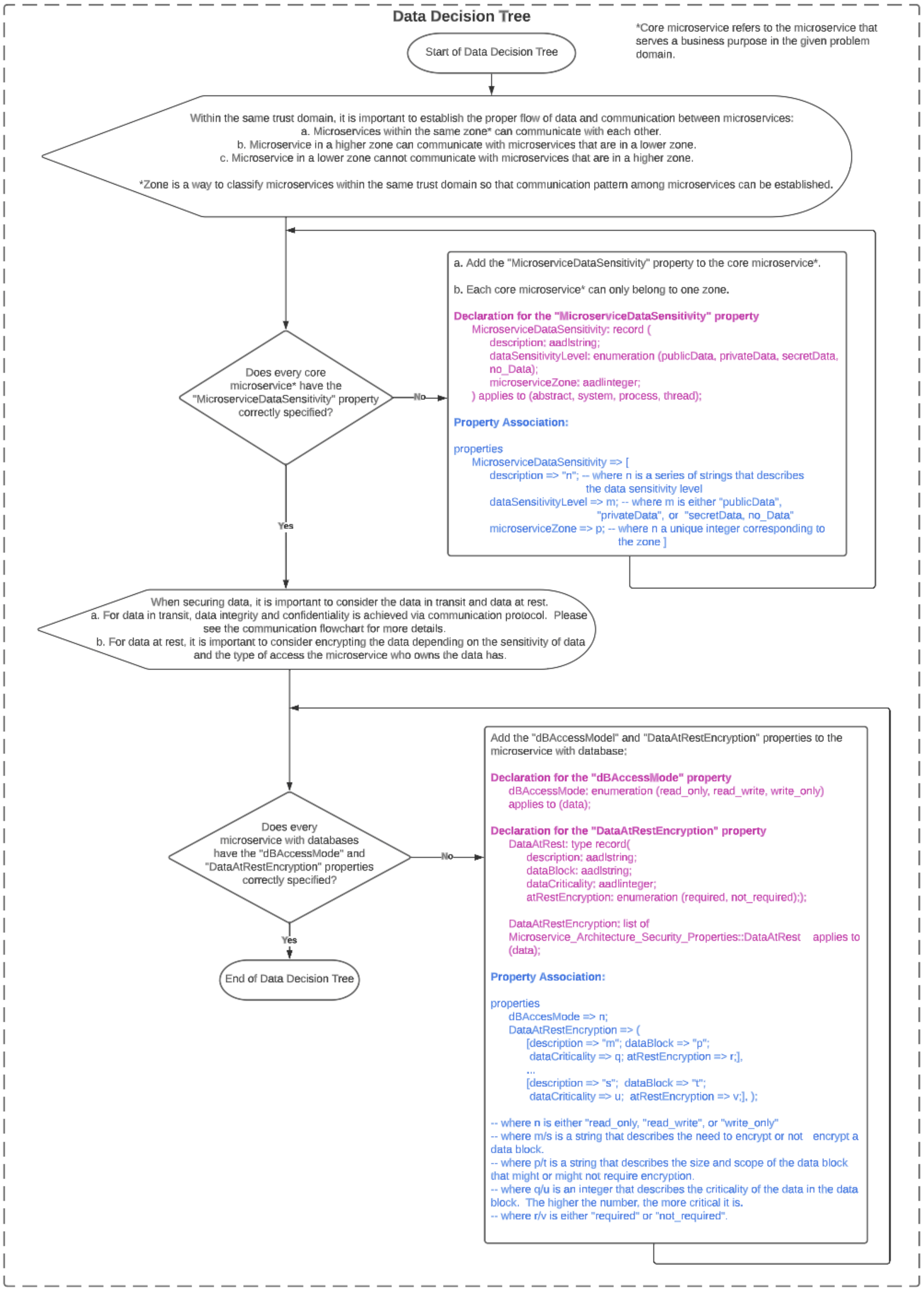
```
    patch_type: type record (
        softwareName: aadlstring;
        frequency: aadlstring;
    );

    DeploymentType: record (
        description: aadlstring;
        deploymentMechanism: enumeration (physical_Machine, virtual_machine, container,
        platform_as_a_service, function_as_a_service);
        patchList: list of Microservice_Architecture_Security_Properties::patch_type;;
    ) applies to (abstract, system, process, thread);
```

Property Association:

```
properties
    DeploymentType => [
        description => "n";
        deploymentMechanism => m;
        patchList => ([softwareName => p; frequency => q;], ..., [softwareName => a; frequency =>b;]);
     ];

-- where n is a series of strings that describes the deployment mechanism of the microservice
-- where m is either "physical _machine", "virtual_machine", "container", "platform_as_a_service", or
"function_as_a_service"
-- where p/a is a name of the software that needs to be patched and q/b is the frequency in which p/a
requires patching.
```

Figure 25: Deployment and Patching Decision Tree

A.15. DATA DECISION TREE


Data Decision Tree
Start of Data Decision Tree
Does every core microservice* have the "MicroserviceDataSensitivity" property correctly specified?
No
Yes
Does every microservice with databases have the "dBAccessMode" and "DataAtRestEncryption" properties correctly specified?
No
Yes
End of Data Decision Tree

*Core microservice refers to the microservice that serves a business purpose in the given problem domain.

Within the same trust domain, it is important to establish the proper flow of data and communication between microservices:
a. Microservices within the same zone* can communicate with each other.
b. Microservice in a higher zone can communicate with microservices that are in a lower zone.
c. Microservice in a lower zone cannot communicate with microservices that are in a higher zone.

*Zone is a way to classify microservices within the same trust domain so that communication pattern among microservices can be established.

a. Add the "MicroserviceDataSensitivity" property to the core microservice*.

b. Each core microservice* can only belong to one zone.

Declaration for the "MicroserviceDataSensitivity" property

```
    MicroserviceDataSensitivity: record (
        description: aadlstring;
        dataSensitivityLevel: enumeration (publicData, privateData, secretData,
        no_Data);
        microserviceZone: aadlinteger;
    ) applies to (abstract, system, process, thread);
```

Property Association:

```
properties
    MicroserviceDataSensitivity => [
        description => "n"; -- where n is a series of strings that describes
                              the data sensitivity level
        dataSensitivityLevel => m; -- where m is either "publicData",
                                 "privateData", or "secretData, no_Data"
        microserviceZone => p; -- where n a unique integer corresponding to
                                   the zone ]
```

When securing data, it is important to consider the data in transit and data at rest.
a. For data in transit, data integrity and confidentiality is achieved via communication protocol. Please see the communication flowchart for more details.
b. For data at rest, it is important to consider encrypting the data depending on the sensitivity of data and the type of access the microservice who owns the data has.

Add the "dBAccessModel" and "DataAtRestEncryption" properties to the microservice with database:

Declaration for the "dBAccessMode" property

```
    dBAccessMode: enumeration (read_only, read_write, write_only)
    applies to (data);
```

Declaration for the "DataAtRestEncryption" property

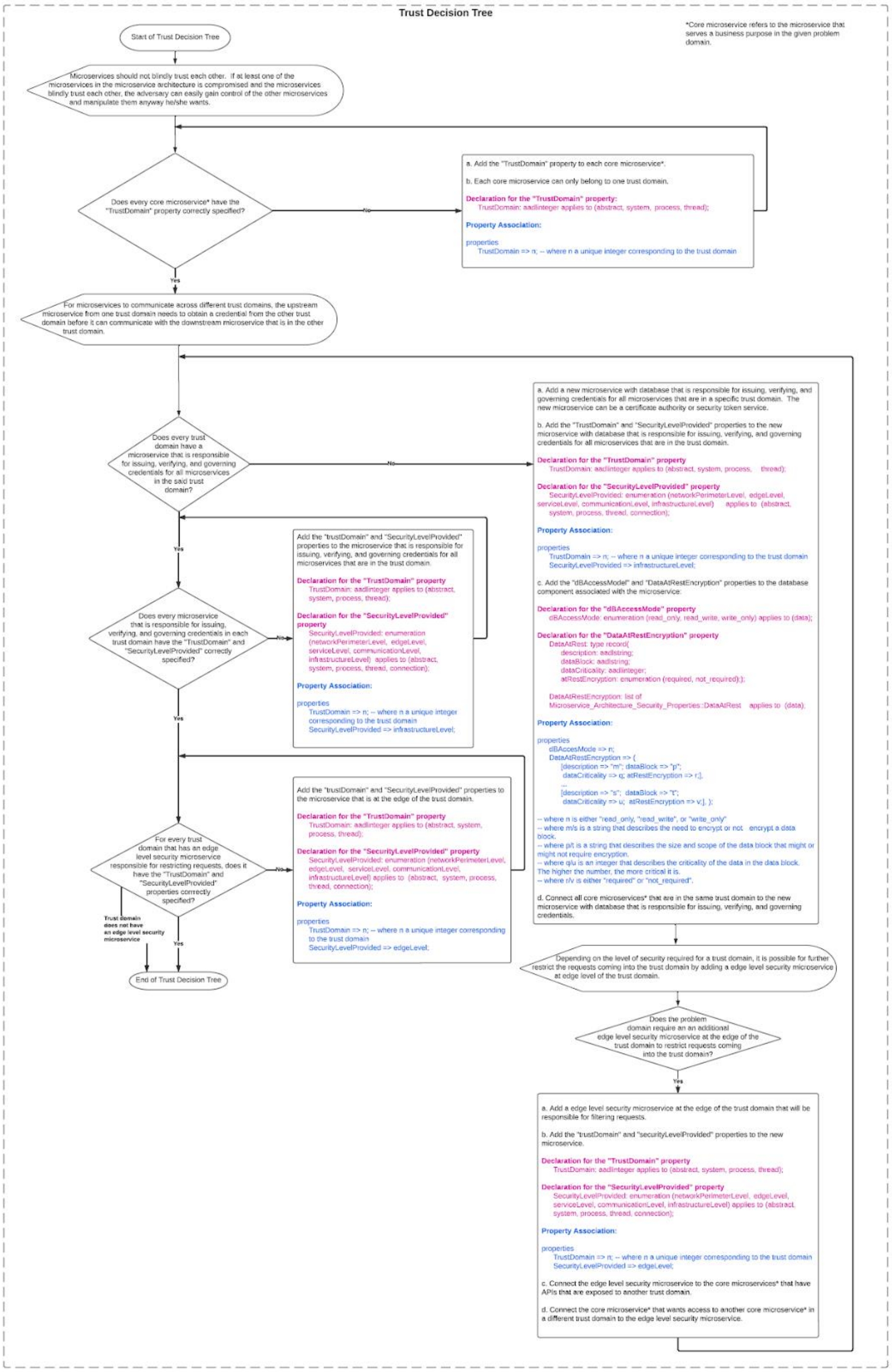
```
    DataAtRest: type record(
        description: aadlstring;
        dataBlock: aadlstring;
        dataCriticality: aadlinteger;
        atRestEncryption: enumeration (required, not_required););

    DataAtRestEncryption: list of
    Microservice_Architecture_Security_Properties::DataAtRest    applies to
    (data);
```

Property Association:

```
properties
    dBAccesMode => n;
    DataAtRestEncryption => (
        [description => "m"; dataBlock => "p";
         dataCriticality => q; atRestEncryption => r;],
        ...
        [description => "s";  dataBlock => "t";
         dataCriticality => u;  atRestEncryption => v;], );

-- where n is either "read_only, "read_write", or "write_only"
-- where m/s is a string that describes the need to encrypt or not   encrypt a
data block.
-- where p/t is a string that describes the size and scope of the data block
that might or might not require encryption.
-- where q/u is an integer that describes the criticality of the data in the data
block.  The higher the number, the more critical it is.
-- where r/v is either "required" or "not_required".
```

Figure 26: Data Decision Tree

A.16. Trust Decision Tree



Trust Decision Tree
Start of Trust Decision Tree
No
Yes
No
Yes
No
Yes
No
Trust domain does not have an edge level security microservice
Yes
End of Trust Decision Tree
Yes

*Core microservice refers to the microservice that serves a business purpose in the given problem domain.

Microservices should not blindly trust each other. If at least one of the microservices in the microservice architecture is compromised and the microservices blindly trust each other, the adversary can easily gain control of the other microservices and manipulate them anyway he/she wants.

Does every core microservice* have the "TrustDomain" property correctly specified?

a. Add the "TrustDomain" property to each core microservice*.

b. Each core microservice can only belong to one trust domain.

```
Declaration for the "TrustDomain" property:
    TrustDomain: aadlinteger applies to (abstract, system, process, thread);

Property Association:

properties
    TrustDomain => n; -- where n a unique integer corresponding to the trust domain
```

For microservices to communicate across different trust domains, the upstream microservice from one trust domain needs to obtain a credential from the other trust domain before it can communicate with the downstream microservice that is in the other trust domain.

Does every trust domain have a microservice that is responsible for issuing, verifying, and governing credentials for all microservices in the said trust domain?

a. Add a new microservice with database that is responsible for issuing, verifying, and governing credentials for all microservices that are in a specific trust domain. The new microservice can be a certificate authority or security token service.

b. Add the "TrustDomain" and "SecurityLevelProvided" properties to the new microservice with database that is responsible for issuing, verifying, and governing credentials for all microservices that are in the trust domain.

```
Declaration for the "TrustDomain" property
    TrustDomain: aadlinteger applies to (abstract, system, process,    thread);

Declaration for the "SecurityLevelProvided" property
    SecurityLevelProvided: enumeration (networkPerimeterLevel, edgeLevel,
serviceLevel, communicationLevel, infrastructureLevel)     applies to  (abstract,
    system, process, thread, connection);

Property Association:

properties
    TrustDomain => n; -- where n a unique integer corresponding to the trust domain
    SecurityLevelProvided => infrastructureLevel;
```

c. Add the "dBAccessModel" and "DataAtRestEncryption" properties to the database component associated with the microservice:

```
Declaration for the "dBAccessMode" property
    dBAccessMode: enumeration (read_only, read_write, write_only) applies to (data);

Declaration for the "DataAtRestEncryption" property
    DataAtRest: type record(
        description: aadlstring;
        dataBlock: aadlstring;
        dataCriticality: aadlinteger;
        atRestEncryption: enumeration (required, not_required););

    DataAtRestEncryption: list of
    Microservice_Architecture_Security_Properties::DataAtRest    applies to  (data);

Property Association:

properties
    dBAccesMode => n;
    DataAtRestEncryption => (
        [description => "m"; dataBlock => "p";
         dataCriticality => q; atRestEncryption => r;],
        ...
        [description => "s";  dataBlock => "t";
         dataCriticality => u;  atRestEncryption => v;], );

-- where n is either "read_only, "read_write", or "write_only"
-- where m/s is a string that describes the need to encrypt or not   encrypt a data
block.
-- where p/t is a string that describes the size and scope of the data block that might or
might not require encryption.
-- where q/u is an integer that describes the criticality of the data in the data block.
The higher the number, the more critical it is.
-- where r/v is either "required" or "not_required".
```

d. Connect all core microservices* that are in the same trust domain to the new microservice with database that is responsible for issuing, verifying, and governing credentials.

Depending on the level of security required for a trust domain, it is possible for further restrict the requests coming into the trust domain by adding a edge level security microservice at edge level of the trust domain.

Does the problem domain require an an additional edge level security microservice at the edge of the trust domain to restrict requests coming into the trust domain?

a. Add a edge level security microservice at the edge of the trust domain that will be responsible for filtering requests.

b. Add the "trustDomain" and "securityLevelProvided" properties to the new microservice.

```
Declaration for the "TrustDomain" property
    TrustDomain: aadlinteger applies to (abstract, system, process, thread);

Declaration for the "SecurityLevelProvided" property
    SecurityLevelProvided: enumeration (networkPerimeterLevel, edgeLevel,
    serviceLevel, communicationLevel, infrastructureLevel) applies to (abstract,
    system, process, thread, connection);

Property Association:

properties
    TrustDomain => n; -- where n a unique integer corresponding to the trust domain
    SecurityLevelProvided => edgeLevel;
```

c. Connect the edge level security microservice to the core microservices* that have APIs that are exposed to another trust domain.

d. Connect the core microservice* that wants access to another core microservice* in a different trust domain to the edge level security microservice.

Does every microservice that is responsible for issuing, verifying, and governing credentials in each trust domain have the "TrustDomain" and "SecurityLevelProvided" correctly specified?

Add the "trustDomain" and "SecurityLevelProvided" properties to the microservice that is responsible for issuing, verifying, and governing credentials for all microservices that are in the trust domain.

```
Declaration for the "TrustDomain" property
    TrustDomain: aadlinteger applies to (abstract,
    system, process, thread);

Declaration for the "SecurityLevelProvided"
property
    SecurityLevelProvided: enumeration
    (networkPerimeterLevel,  edgeLevel,
    serviceLevel, communicationLevel,
    infrastructureLevel)  applies to (abstract,
    system, process, thread, connection);

Property Association:

properties
    TrustDomain => n; -- where n a unique integer
    corresponding to the trust domain
    SecurityLevelProvided => infrastructureLevel;
```

For every trust domain that has an edge level security microservice responsible for restricting requests, does it have the "TrustDomain" and "SecurityLevelProvided" properties correctly specified?

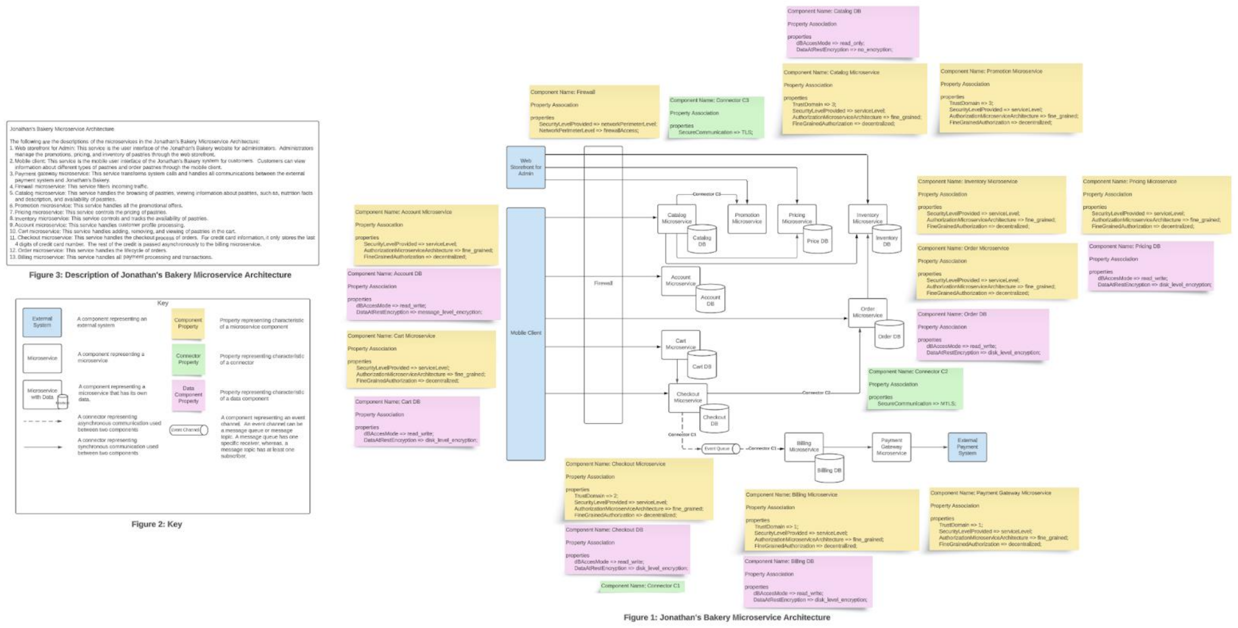
Add the "trustDomain" and "SecurityLevelProvided" properties to the microservice that is at the edge of the trust domain.

```
Declaration for the "TrustDomain" property
    TrustDomain: aadlinteger applies to (abstract, system,
    process, thread);

Declaration for the "SecurityLevelProvided" property
    SecurityLevelProvided: enumeration (networkPerimeterLevel,
    edgeLevel,  serviceLevel, communicationLevel,
    infrastructureLevel) applies to  (abstract,  system, process,
    thread, connection);

Property Association:

properties
    TrustDomain => n; -- where n a unique integer corresponding
    to the trust domain
    SecurityLevelProvided => edgeLevel;
```

Figure 27: Trust Decision Tree

Appendix B: Survey Questions

This section presents the microservice architecture, general instructions and questions included in the survey.



Jonathan's Bakery Microservice Architecture

The following are the descriptions of the microservices in the Jonathan's Bakery Microservice Architecture:

1. Web storefront for Admin: This service is the user interface of the Jonathan's Bakery website for administrators. Administrators manage the promotions, pricing, and inventory of pastries through the web storefront.
2. Mobile client: This service is the mobile user interface of the Jonathan's Bakery system for customers. Customers can view information about different types of pastries and order pastries through the mobile client.
3. Payment gateway microservice: This service transforms system calls and handles all communications between the external payment system and Jonathan's Bakery.
4. Firewall microservice: This service filters incoming traffic.
5. Catalog microservice: This service handles the browsing of pastries, viewing information about pastries, such as, nutrition facts and description, and availability of pastries.
6. Promotion microservice: This service handles all the promotional offers.
7. Pricing microservice: This service controls the pricing of pastries.
8. Inventory microservice: This service controls and tracks the availability of pastries.
9. Account microservice: This service handles customer profile processing.
10. Cart microservice: This service handles adding, removing, and viewing of pastries in the cart.
11. Checkout microservice: This service handles the checkout process of orders. For credit card information, it only stores the last 4 digits of credit card number. The rest of the credit is passed asynchronously to the billing microservice.
12. Order microservice: This service handles the lifecycle of orders.
13. Billing microservice: This service handles all payment processing and transactions.

Figure 3: Description of Jonathan's Bakery Microservice Architecture

External System	A component representing an external system	Component Property	Property representing characteristic of a microservice component
Microservice	A component representing a microservice	Connector Property	Property representing characteristic of a connector
Microservice with Data	A component representing a microservice that has its own data.	Data Component Property	Property representing characteristic of a data component
- - - - →	A connector representing asynchronous communication used between two components	Event Channel	A component representing an event channel. An event channel can be a message queue or message topic. A message queue has one specific receiver, whereas, a message topic has at least one subscriber.
——→	A connector representing synchronous communication used between two components		

Figure 2: Key

```
Component Name: Catalog DB
Property Association
properties
    dBAccesMode => read_only;
    DataAtRestEncryption => no_encryption;

Component Name: Firewall
Property Assocation
properties
    SecurityLevelProvided => networkPerimeterLevel;
    NetworkPerimeterLevel => firewallAccess;

Component Name: Connector C3
Property Association
properties
    SecureCommunication => TLS;

Component Name: Catalog Microservice
Property Association
properties
    TrustDomain => 3;
    SecurityLevelProvided => serviceLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => decentralized;

Component Name: Promotion Microservice
Property Association
properties
    TrustDomain => 3;
    SecurityLevelProvided => serviceLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => decentralized;

Component Name: Inventory Microservice
Property Association
properties
    SecurityLevelProvided => serviceLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => decentralized;

Component Name: Pricing Microservice
Property Association
properties
    SecurityLevelProvided => serviceLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => decentralized;

Component Name: Account Microservice
Property Assocation
properties
    SecurityLevelProvided => serviceLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => decentralized;

Component Name: Order Microservice
Property Association
properties
    SecurityLevelProvided => serviceLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => decentralized;

Component Name: Pricing DB
Property Association
properties
    dBAccesMode => read_write;
    DataAtRestEncryption => disk_level_encryption;

Component Name: Account DB
Property Association
properties
    dBAccesMode => read_write;
    DataAtRestEncryption => message_level_encryption;

Component Name: Order DB
Property Association
properties
    dBAccessMode => read_write;
    DataAtRestEncryption => disk_level_encryption;

Component Name: Cart Microservice
Property Association
properties
    SecurityLevelProvided => serviceLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => decentralized;

Component Name: Connector C2
Property Association
properties
    SecureCommunication => MTLS;

Component Name: Cart DB
Property Association
properties
    dBAccesMode => read_write;
    DataAtRestEncryption => disk_level_encryption;

Component Name: Checkout Microservice
Property Association
properties
    TrustDomain => 2;
    SecurityLevelProvided => serviceLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => decentralized;

Component Name: Billing Microservice
Property Association
properties
    TrustDomain => 1;
    SecurityLevelProvided => serviceLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => decentralized;

Component Name: Payment Gateway Microservice
Property Association
properties
    TrustDomain => 1;
    SecurityLevelProvided => serviceLevel;
    AuthorizationMicroserviceArchitecture => fine_grained;
    FineGrainedAuthorization => decentralized;

Component Name: Checkout DB
Property Association
properties
    dBAccesMode => read_write;
    DataAtRestEncryption => disk_level_encryption;

Component Name: Billing DB
Property Association
properties
    dBAccesMode => read_write;
    DataAtRestEncryption => disk_level_encryption;

Component Name: Connector C1
```

Figure 1: Jonathan's Bakery Microservice Architecture

Figure 28: Microservice Architecture

General instructions:

1. Each question below presents a unique set of facts (optional) and requirements.
2. Please use the microservice architecture provided in Figure 1 to respond to each question. Link to the microservice architecture: https://minersutep-my.sharepoint.com/:b:/g/personal/wyetai_utep_edu/Ed5lqgxGKARLsa0VlgzqwgoBH9cMMlORaAJRvpd27jJ3cg?e=IO9efp
3. Note that each question does not use any information from the previous question.

1. Please enter your group number. *

Enter your answer

2. There should be a total of 3 members per group (some groups will have 4 members per group). Please state the first and last names of all your group members. *

Enter your answer

3. **Requirements:**

    1. Checkout microservice and Order microservice belong to the same trust domain.
    2. Billing microservice belongs to a different trust domain.
    3. In order for a customer to pay for the purchase, Checkout microservice needs to communicate with Order microservice and Billing microservice.

    Instructions:

    1. Referring to the microservice architecture presented in figure 1, describe any changes that you would make to the components (and associated properties) and connectors (and associated properties) to satisfy the requirements.
    2. Justify your answer.

*

4. **Facts:**

    1. Development Team A will be in charge of the development of Account microservice.
    2. Development Team B will be in charge of the development of Inventory microservice.
    3. Development Team A and Development Team B use different technology stacks.

    Requirements:

    1. Account microservice will be running on a virtual machine. The operating system in the virtual machine will require an update every two weeks.
    2. Data stored in Account microservice will require encryption at the disk level.
    3. Inventory microservice will be running on AWS.
    4. Data stored in Inventory microservice will require encryption at the disk level.
    5. Account and Inventory microservices will be developed from scratch without using any third party libraries.

    Instructions:

    1. Referring to the microservice architecture presented in figure 1, describe any changes that you would make to the components (and associated properties) and connectors (and associated properties) to satisfy the requirements.
    2. Justify your answer.

*

5. **Requirements**

    1. Catalog microservice and Promotion microservice belong to the same trust domain.
    2. Catalog microservice makes a request to Promotion microservice.
    3. Catalog microservice needs to verify the identity of Promotion microservice to make sure it is communicating with an authentic service.
    4. Promotion microservice does not need to verify the identity of Catalog microservice.

    Instructions:

    1. Referring to the microservice architecture presented in figure 1, describe any changes that you would make to the components (and associated properties) and connectors (and associated properties) to satisfy the requirements.
    2. Justify your answer.

    *

6. **Requirements:**

    1. The Mobile client interacts with Catalog microservice.
    2. Catalog microservice has two APIs: Provide_Pastry_Nutrition_Information and Update_Pastry_Availability.
    3. Provide_Pastry_Nutrition_Information is a public API and can handle a maximum of 20 requests per second.
    4. Update_Pastry_Availability is an API that is only accessible by internal microservices and can handle a maximum of 5 requests per second.
    5. The mobile client should not send more than 2GB of data to Catalog microservice.
    6. The mobile client should not send more than 10 requests per second to the Catalog microservice. If the rate of request is more than 10 requests per second, the system will wait for 5 mins before processing anymore requests from the mobile client.
    7. If the rate of request exceeds 15 requests per second, the system will error out and reject all requests to the Catalog microservice from the mobile client.

    Instructions:

    1. Referring to the microservice architecture presented in figure 1, describe any changes that you would make to the components (and associated properties) and connectors (and associated properties) to satisfy the requirements.
    2. Justify your answer.

    *

7. **Fact:**

    1. The microservice architecture currently supports decentralized authorization mechanism.

    Requirement:

    1. The microservice architecture needs to be updated to support centralized authorization mechanism with latency being one of the constraints.

    Instructions:

    1. Referring to the microservice architecture presented in figure 1, describe any changes that you would make to the components (and associated properties) and connectors (and associated properties) to satisfy the requirements.
    2. Justify your answer.

    *

8. **Requirements:**

    1. Billing microservice stores credit transaction data.
    2. Payment Gateway microservice acts an adapter to facilitate communications between the external third party payment system and Billing microservice.
    3. The microservice architecture needs to guarantee that Billing Microservice shall never accept communication from another microservice with the same trust domain (current or future) that does not share the same data sensitivity level.

    Instructions:

    1. Referring to the microservice architecture presented in figure 1, describe any changes that you would make to the components (and associated properties) and connectors (and associated properties) to satisfy the requirements.
    2. Justify your answer.

    *

9. **Fact:**

    1. According to the defense in depth principle, security should be applied and layered throughout the microservice architecture in order to protect the system, data, and information.

    Requirement:

    1. The company wants to strengthen the level of security at the network level.

    Instructions:

    1. Referring to the microservice architecture presented in figure 1, describe any changes that you would make to the components (and associated properties) and connectors (and associated properties) to satisfy the requirements.
    2. Justify your answer.

    *

10. **Requirements:**

    1. The company wants an authentication mechanism that allows carrying of user identity from external applications to microservices and between microservices.
    2. The support of disk level encryption of data is required.

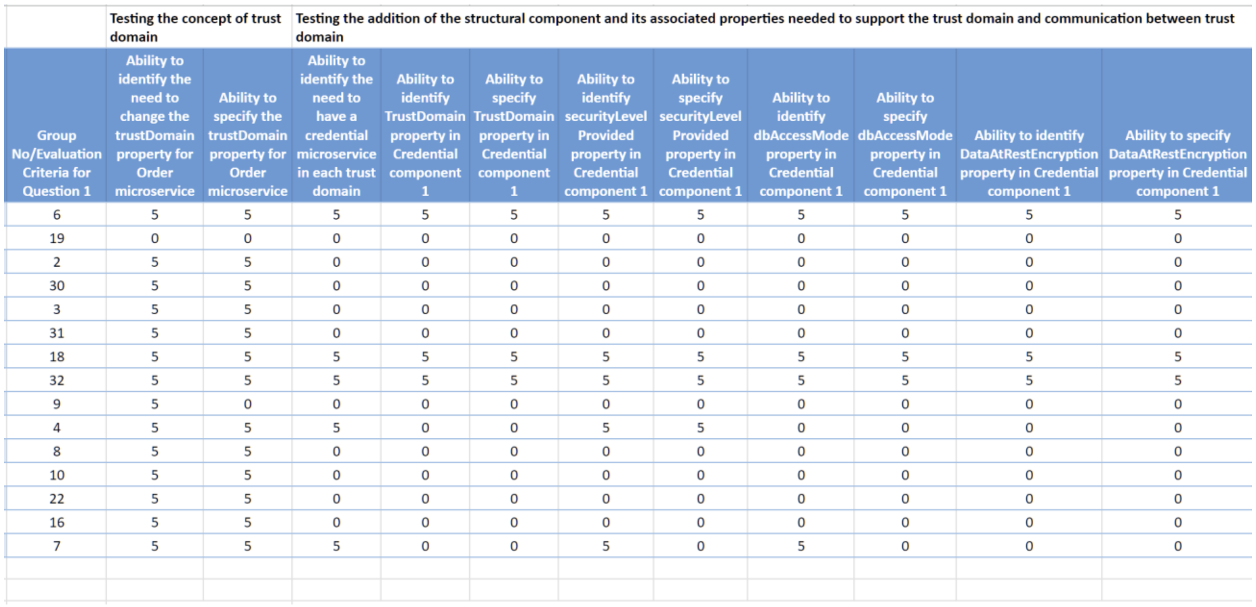
    Instructions:

    1. Referring to the microservice architecture presented in figure 1, describe any changes that you would make to the components (and associated properties) and connectors (and associated properties) to satisfy the requirements.
    2. Justify your answer.

    *

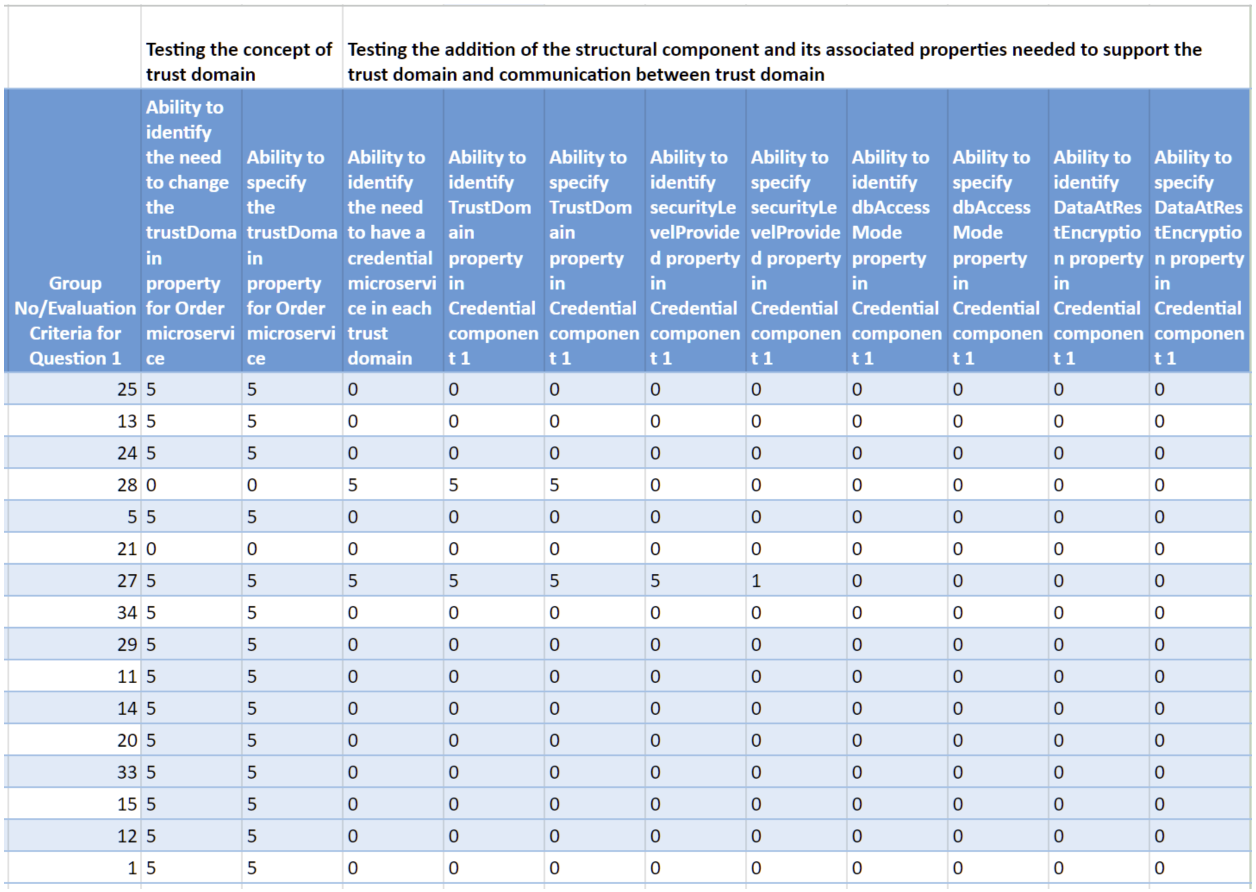
Appendix C: Survey Results

Appendix C presents the scores received by each team for the survey on a per question basis.



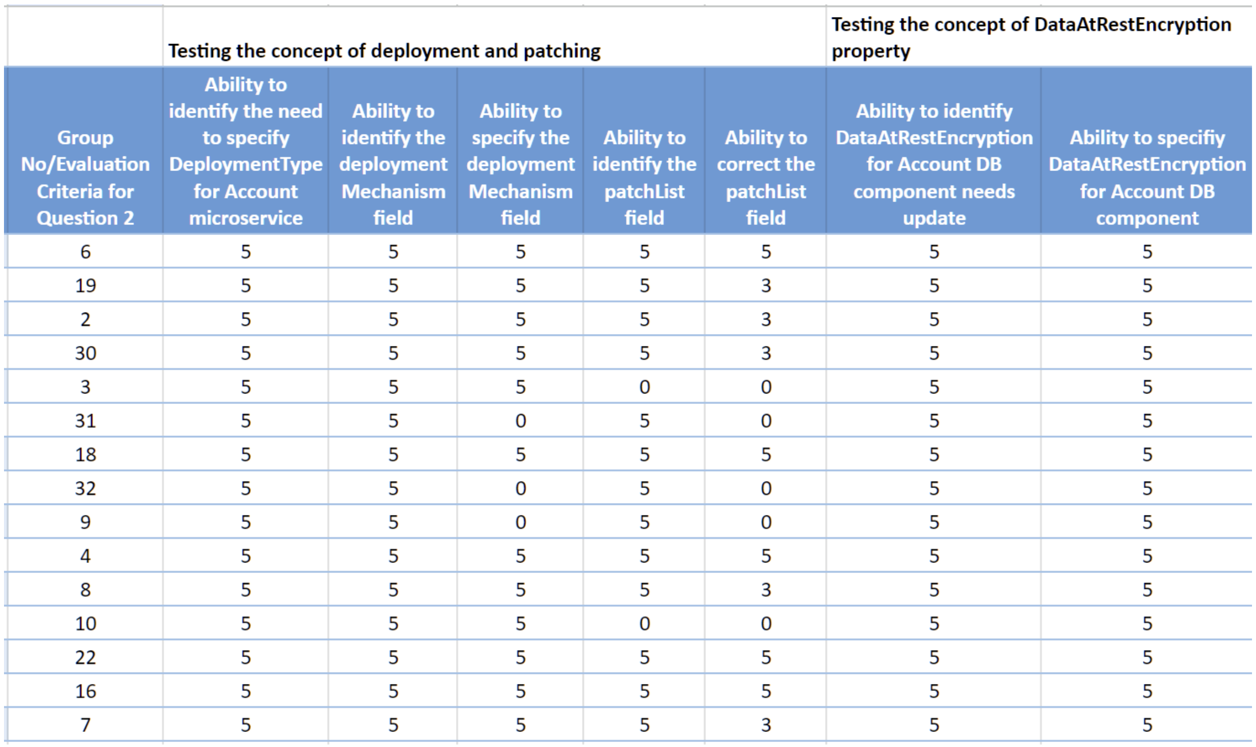
	Testing the concept of trust domain		Testing the addition of the structural component and its associated properties needed to support the trust domain and communication between trust domain								
Group No/Evaluation Criteria for Question 1	Ability to identify the need to change the trustDomain property for Order microservice	Ability to specify the trustDomain property for Order microservice	Ability to identify the need to have a credential microservice in each trust domain	Ability to identify TrustDomain property in Credential component 1	Ability to specify TrustDomain property in Credential component 1	Ability to identify securityLevel Provided property in Credential component 1	Ability to specify securityLevel Provided property in Credential component 1	Ability to identify dbAccessMode property in Credential component 1	Ability to specify dbAccessMode property in Credential component 1	Ability to identify DataAtRestEncryption property in Credential component 1	Ability to specify DataAtRestEncryption property in Credential component 1
6	5	5	5	5	5	5	5	5	5	5	5
19	0	0	0	0	0	0	0	0	0	0	0
2	5	5	0	0	0	0	0	0	0	0	0
30	5	5	0	0	0	0	0	0	0	0	0
3	5	5	0	0	0	0	0	0	0	0	0
31	5	5	0	0	0	0	0	0	0	0	0
18	5	5	5	5	5	5	5	5	5	5	5
32	5	5	5	5	5	5	5	5	5	5	5
9	5	0	0	0	0	0	0	0	0	0	0
4	5	5	5	0	0	5	5	0	0	0	0
8	5	5	0	0	0	0	0	0	0	0	0
10	5	5	0	0	0	0	0	0	0	0	0
22	5	5	0	0	0	0	0	0	0	0	0
16	5	5	0	0	0	0	0	0	0	0	0
7	5	5	5	0	0	5	0	5	0	0	0

Figure 29: Question 1 Scores for Treatment Groups



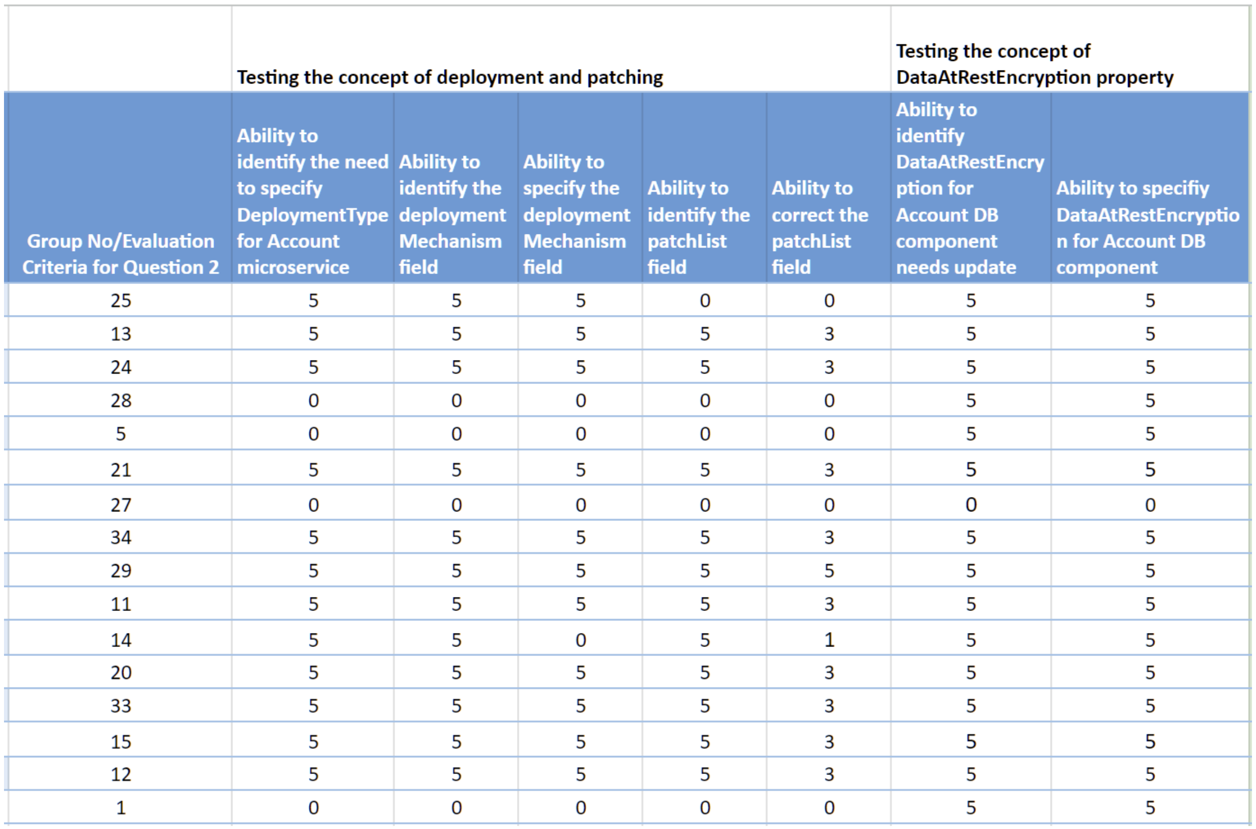
	Testing the concept of trust domain		Testing the addition of the structural component and its associated properties needed to support the trust domain and communication between trust domain								
Group No/Evaluation Criteria for Question 1	Ability to identify the need to change the trustDomain property for Order microservice	Ability to specify the trustDomain property for Order microservice	Ability to identify the need to have a credential microservice in each trust domain	Ability to identify TrustDomain property in Credential component 1	Ability to specify TrustDomain property in Credential component 1	Ability to identify securityLevelProvided property in Credential component 1	Ability to specify securityLevelProvided property in Credential component 1	Ability to identify dbAccess Mode property in Credential component 1	Ability to specify dbAccess Mode property in Credential component 1	Ability to identify DataAtRestEncryption property in Credential component 1	Ability to specify DataAtRestEncryption property in Credential component 1
25	5	5	0	0	0	0	0	0	0	0	0
13	5	5	0	0	0	0	0	0	0	0	0
24	5	5	0	0	0	0	0	0	0	0	0
28	0	0	5	5	5	0	0	0	0	0	0
5	5	5	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0
27	5	5	5	5	5	5	1	0	0	0	0
34	5	5	0	0	0	0	0	0	0	0	0
29	5	5	0	0	0	0	0	0	0	0	0
11	5	5	0	0	0	0	0	0	0	0	0
14	5	5	0	0	0	0	0	0	0	0	0
20	5	5	0	0	0	0	0	0	0	0	0
33	5	5	0	0	0	0	0	0	0	0	0
15	5	5	0	0	0	0	0	0	0	0	0
12	5	5	0	0	0	0	0	0	0	0	0
1	5	5	0	0	0	0	0	0	0	0	0

Figure 30: Question 1 Scores for Control Groups



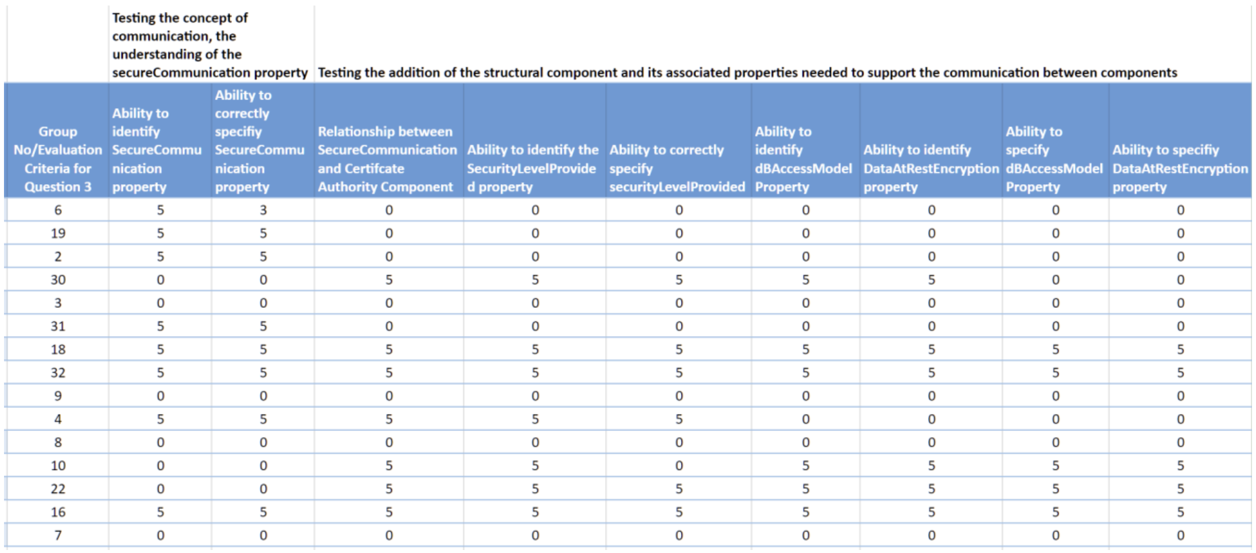
	Testing the concept of deployment and patching					Testing the concept of DataAtRestEncryption property	
Group No/Evaluation Criteria for Question 2	Ability to identify the need to specify DeploymentType for Account microservice	Ability to identify the deployment Mechanism field	Ability to specify the deployment Mechanism field	Ability to identify the patchList field	Ability to correct the patchList field	Ability to identify DataAtRestEncryption for Account DB component needs update	Ability to specifiy DataAtRestEncryption for Account DB component
6	5	5	5	5	5	5	5
19	5	5	5	5	3	5	5
2	5	5	5	5	3	5	5
30	5	5	5	5	3	5	5
3	5	5	5	0	0	5	5
31	5	5	0	5	0	5	5
18	5	5	5	5	5	5	5
32	5	5	0	5	0	5	5
9	5	5	0	5	0	5	5
4	5	5	5	5	5	5	5
8	5	5	5	5	3	5	5
10	5	5	5	0	0	5	5
22	5	5	5	5	5	5	5
16	5	5	5	5	5	5	5
7	5	5	5	5	3	5	5

Figure 31: Question 2 Scores for Treatment Groups



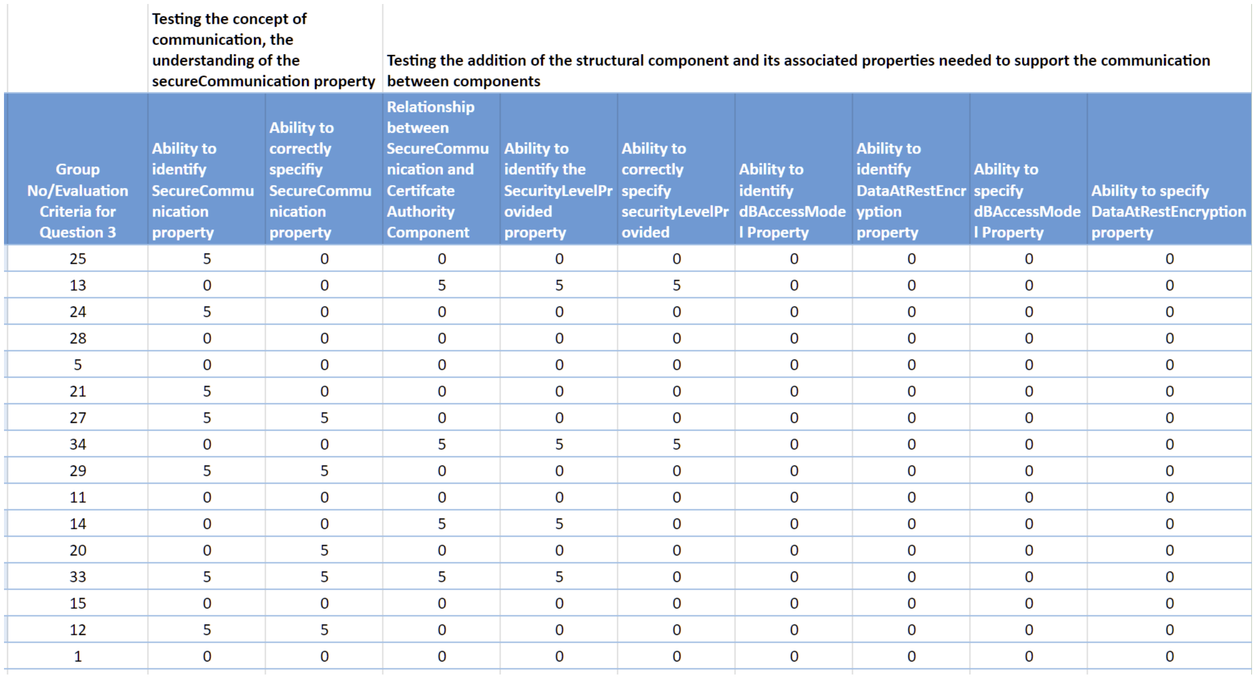
	Testing the concept of deployment and patching					Testing the concept of DataAtRestEncryption property	
Group No/Evaluation Criteria for Question 2	Ability to identify the need to specify DeploymentType for Account microservice	Ability to identify the deployment Mechanism field	Ability to specify the deployment Mechanism field	Ability to identify the patchList field	Ability to correct the patchList field	Ability to identify DataAtRestEncryption for Account DB component needs update	Ability to specifiy DataAtRestEncryption for Account DB component
25	5	5	5	0	0	5	5
13	5	5	5	5	3	5	5
24	5	5	5	5	3	5	5
28	0	0	0	0	0	5	5
5	0	0	0	0	0	5	5
21	5	5	5	5	3	5	5
27	0	0	0	0	0	0	0
34	5	5	5	5	3	5	5
29	5	5	5	5	5	5	5
11	5	5	5	5	3	5	5
14	5	5	0	5	1	5	5
20	5	5	5	5	3	5	5
33	5	5	5	5	3	5	5
15	5	5	5	5	3	5	5
12	5	5	5	5	3	5	5
1	0	0	0	0	0	5	5

Figure 32: Question 2 Scores for Control Groups



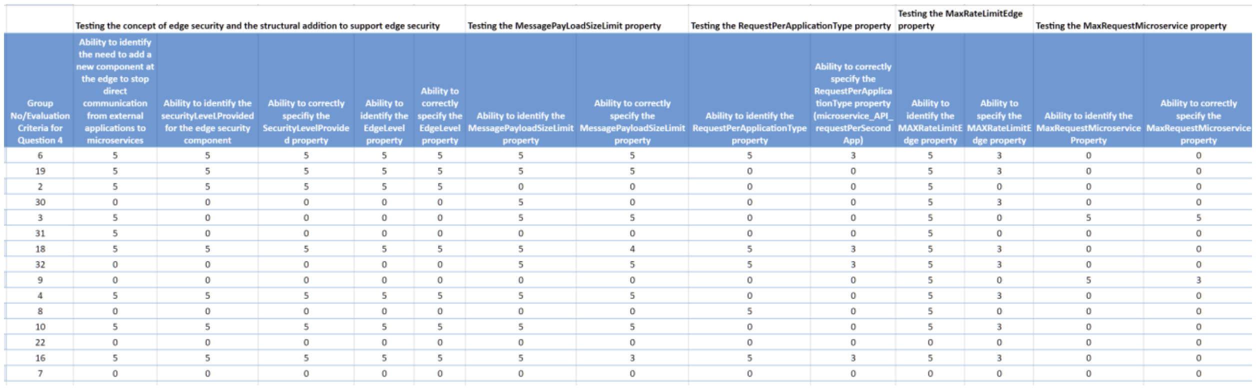
	Testing the concept of communication, the understanding of the secureCommunication property		Testing the addition of the structural component and its associated properties needed to support the communication between components						
Group No/Evaluation Criteria for Question 3	Ability to identify SecureCommunication property	Ability to correctly specifiy SecureCommunication property	Relationship between SecureCommunication and Certifcate Authority Component	Ability to identify the SecurityLevelProvided property	Ability to correctly specify securityLevelProvided	Ability to identify dBAccessModel Property	Ability to identify DataAtRestEncryption property	Ability to specify dBAccessModel Property	Ability to specifiy DataAtRestEncryption property
6	5	3	0	0	0	0	0	0	0
19	5	5	0	0	0	0	0	0	0
2	5	5	0	0	0	0	0	0	0
30	0	0	5	5	5	5	5	0	0
3	0	0	0	0	0	0	0	0	0
31	5	5	0	0	0	0	0	0	0
18	5	5	5	5	5	5	5	5	5
32	5	5	5	5	5	5	5	5	5
9	0	0	0	0	0	0	0	0	0
4	5	5	5	5	5	0	0	0	0
8	0	0	0	0	0	0	0	0	0
10	0	0	5	5	0	5	5	5	5
22	0	0	5	5	5	5	5	5	5
16	5	5	5	5	5	5	5	5	5
7	0	0	0	0	0	0	0	0	0

Figure 33: Question 3 Scores for Treatment Groups



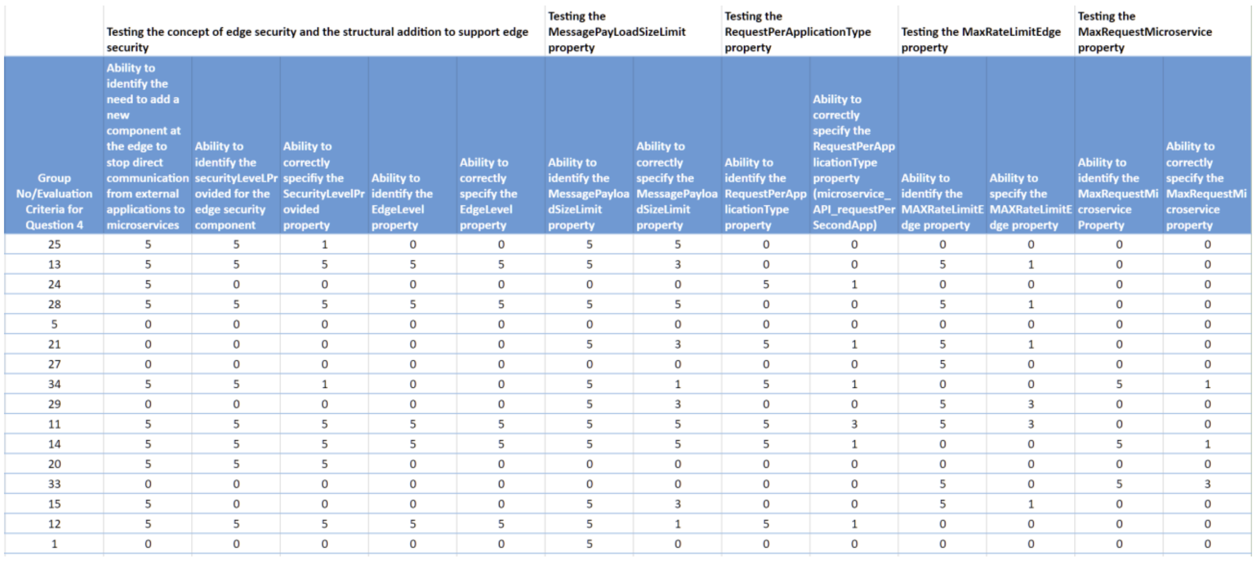
	Testing the concept of communication, the understanding of the secureCommunication property		Testing the addition of the structural component and its associated properties needed to support the communication between components						
Group No/Evaluation Criteria for Question 3	Ability to identify SecureCommunication property	Ability to correctly specifiy SecureCommunication property	Relationship between SecureCommunication and Certifcate Authority Component	Ability to identify the SecurityLevelProvided property	Ability to correctly specify securityLevelProvided	Ability to identify dBAccessModel Property	Ability to identify DataAtRestEncryption property	Ability to specify dBAccessModel Property	Ability to specify DataAtRestEncryption property
25	5	0	0	0	0	0	0	0	0
13	0	0	5	5	5	0	0	0	0
24	5	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
21	5	0	0	0	0	0	0	0	0
27	5	5	0	0	0	0	0	0	0
34	0	0	5	5	5	0	0	0	0
29	5	5	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0
14	0	0	5	5	0	0	0	0	0
20	0	5	0	0	0	0	0	0	0
33	5	5	5	5	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0
12	5	5	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0

Figure 34: Question 3 Scores for Control Groups



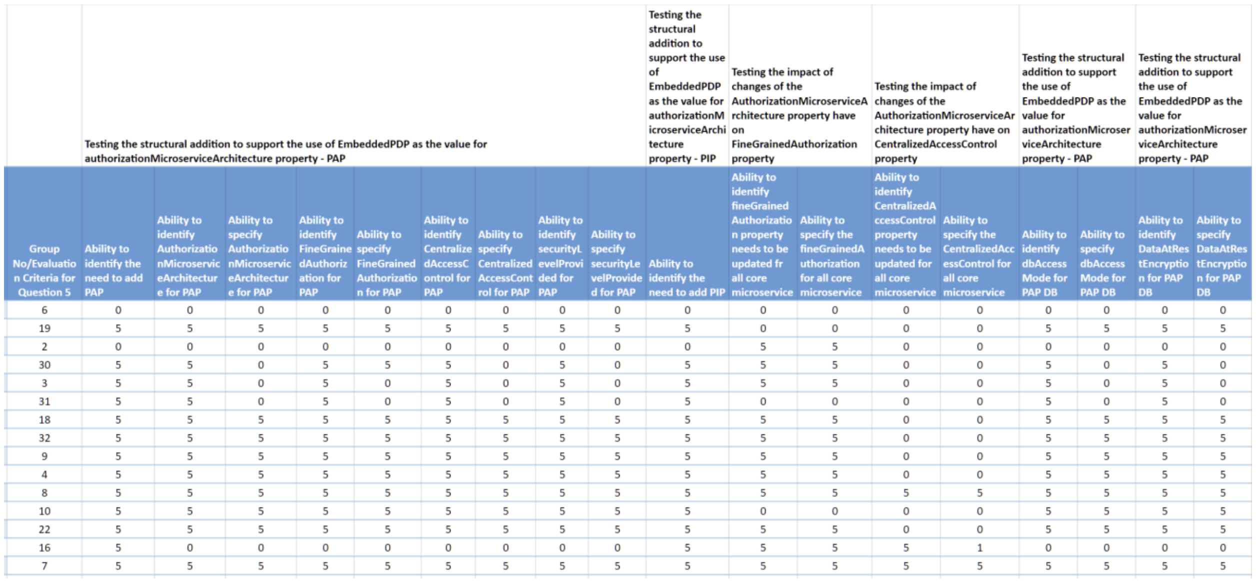
	Testing the concept of edge security and the structural addition to support edge security					Testing the MessagePayLoadSizeLimit property		Testing the RequestPerApplicationType property		Testing the MaxRateLimitEdge property		Testing the MaxRequestMicroservice property	
Group No/Evaluation Criteria for Question 4	Ability to identify the need to add a new component at the edge to stop direct communication from external applications to microservices	Ability to identify the securityLevelProvided for the edge security component	Ability to correctly specifiy the SecurityLevelProvided property	Ability to identify the EdgeLevel property	Ability to correctly specify the EdgeLevel property	Ability to identify the MessagePayloadSizeLimit property	Ability to correctly specify the MessagePayloadSizeLimit property	Ability to identify the RequestPerApplicationType property	Ability to correctly specify the RequestPerApplicationType property (microservice_API_requestPerSecondApp)	Ability to identify the MAXRateLimitEdge property	Ability to specify the MAXRateLimitEdge property	Ability to identify the MaxRequestMicroservice Property	Ability to correctly specify the MaxRequestMicroservice property
6	5	5	5	5	5	5	5	5	3	5	3	0	0
19	5	5	5	5	5	5	5	0	0	5	3	0	0
2	5	5	5	5	5	0	0	0	0	5	0	0	0
30	0	0	0	0	0	5	0	0	0	5	3	0	0
3	5	0	0	0	0	5	5	0	0	5	0	5	5
31	5	0	0	0	0	0	0	0	0	5	0	0	0
18	5	5	5	5	5	5	4	5	3	5	3	0	0
32	0	0	0	0	0	5	5	5	3	5	3	0	0
9	0	0	0	0	0	0	0	0	0	5	0	5	3
4	5	5	5	5	5	5	5	0	0	5	3	0	0
8	0	0	0	0	0	0	0	5	0	5	0	0	0
10	5	5	5	5	5	5	5	0	0	5	3	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0
16	5	5	5	5	5	5	3	5	3	5	3	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35: Question 4 Scores for Treatment Groups



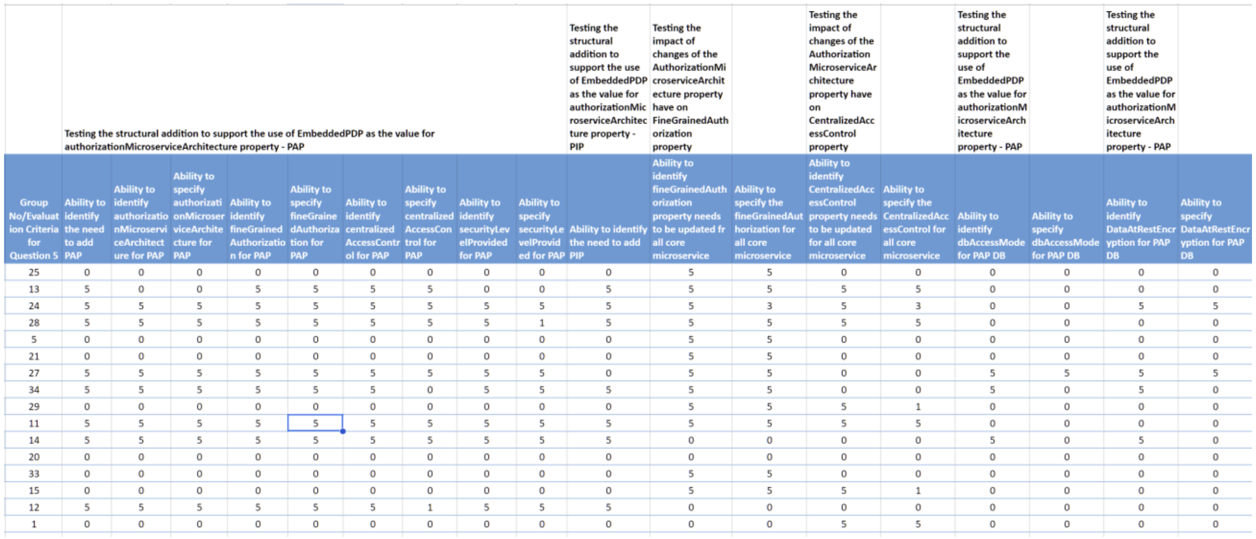
	Testing the concept of edge security and the structural addition to support edge security					Testing the MessagePayLoadSizeLimit property		Testing the RequestPerApplicationType property		Testing the MaxRateLimitEdge property		Testing the MaxRequestMicroservice property	
Group No/Evaluation Criteria for Question 4	Ability to identify the need to add a new component at the edge to stop direct communication from external applications to microservices	Ability to identify the securityLevelProvided for the edge security component	Ability to correctly specifiy the SecurityLevelProvided property	Ability to identify the EdgeLevel property	Ability to correctly specify the EdgeLevel property	Ability to identify the MessagePayloadSizeLimit property	Ability to correctly specify the MessagePayloadSizeLimit property	Ability to identify the RequestPerApplicationType property	Ability to correctly specify the RequestPerApplicationType property (microservice_API_requestPerSecondApp)	Ability to identify the MAXRateLimitEdge property	Ability to specify the MAXRateLimitEdge property	Ability to identify the MaxRequestMicroservice Property	Ability to correctly specify the MaxRequestMicroservice property
25	5	5	1	0	0	5	5	0	0	0	0	0	0
13	5	5	5	5	5	5	3	0	0	5	1	0	0
24	5	0	0	0	0	0	0	5	1	0	0	0	0
28	5	5	5	5	5	5	5	0	0	5	1	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	5	3	5	1	5	1	0	0
27	0	0	0	0	0	0	0	0	0	5	0	0	0
34	5	5	1	0	0	5	1	5	1	0	0	5	1
29	0	0	0	0	0	5	3	0	0	5	3	0	0
11	5	5	5	5	5	5	5	5	3	5	3	0	0
14	5	5	5	5	5	5	5	5	1	0	0	5	1
20	5	5	5	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	5	0	5	3
15	5	0	0	0	0	5	3	0	0	5	1	0	0
12	5	5	5	5	5	5	1	5	1	0	0	0	0
1	0	0	0	0	0	5	0	0	0	0	0	0	0

Figure 36: Question 4 Scores for Control Groups



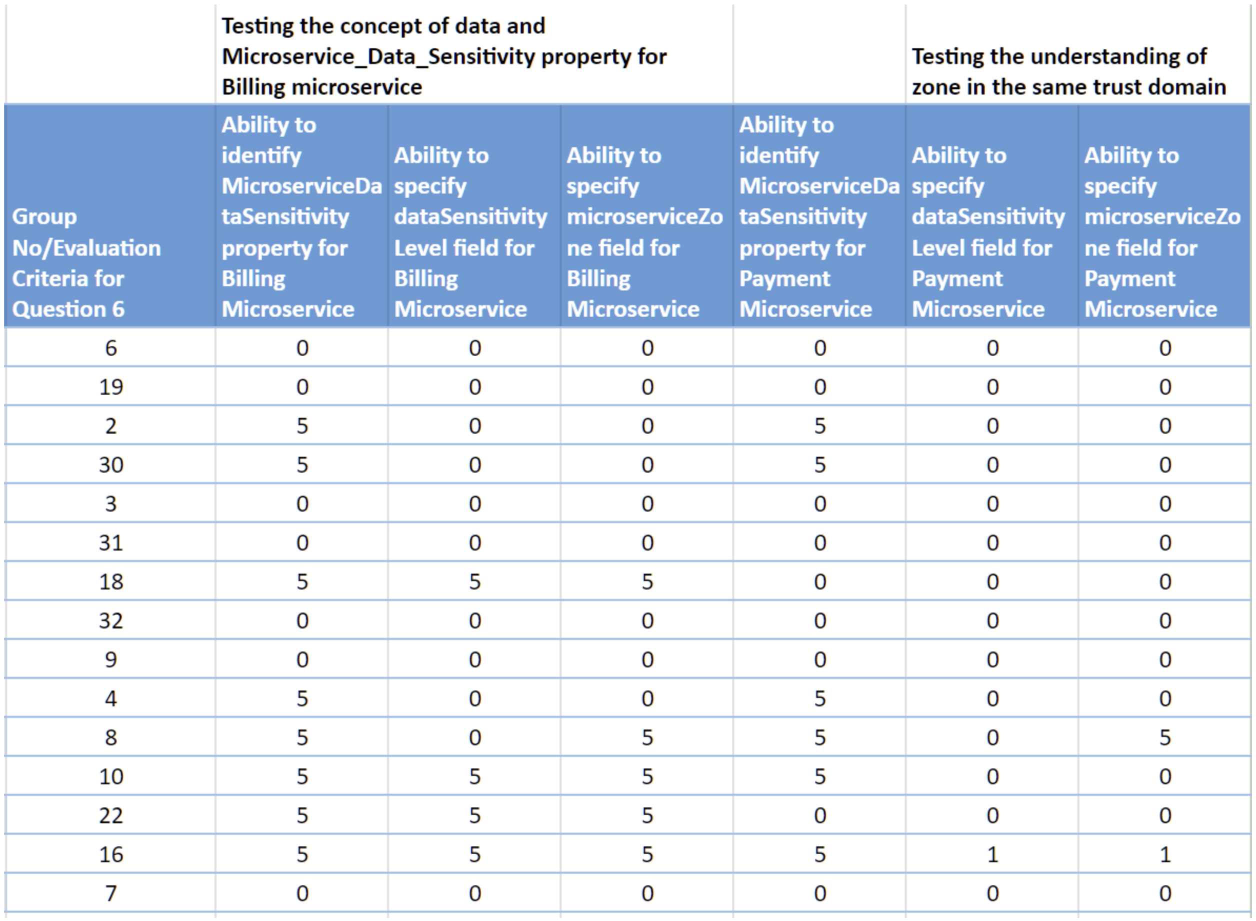
	Testing the structural addition to support the use of EmbeddedPDP as the value for authorizationMicroserviceArchitecture property - PAP									Testing the structural addition to support the use of EmbeddedPDP as the value for authorizationMicroserviceArchitecture property - PIP	Testing the impact of changes of the AuthorizationMicroserviceArchitecture property have on FineGrainedAuthorization property		Testing the impact of changes of the AuthorizationMicroserviceArchitecture property have on CentralizedAccessControl property		Testing the structural addition to support the use of EmbeddedPDP as the value for authorizationMicroserviceArchitecture property - PAP		Testing the structural addition to support the use of EmbeddedPDP as the value for authorizationMicroserviceArchitecture property - PAP	
Group No/Evaluation Criteria for Question 5	Ability to identify the need to add PAP	Ability to identify AuthorizationMicroserviceArchitecture for PAP	Ability to specify AuthorizationMicroserviceArchitecture for PAP	Ability to identify FineGrainedAuthorization for PAP	Ability to specify FineGrainedAuthorization for PAP	Ability to identify CentralizedAccessControl for PAP	Ability to specify CentralizedAccessControl for PAP	Ability to identify securityLevelProvided for PAP	Ability to specify securityLevelProvided for PAP	Ability to identify the need to add PIP	Ability to identify fineGrainedAuthorization property needs to be updated fr all core microservice	Ability to specify the fineGrainedAuthorization for all core microservice	Ability to identify CentralizedAccessControl property needs to be updated for all core microservice	Ability to specify the CentralizedAccessControl for all core microservice	Ability to identify dbAccessMode for PAP DB	Ability to specify dbAccessMode for PAP DB	Ability to identify DataAtRestEncryption for PAP DB	Ability to specify DataAtRestEncryption for PAP DB
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	5	5	5	5	5	5	5	5	5	5	0	0	0	0	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	5	5	0	0	0	0	0	0
30	5	5	0	5	5	5	0	5	0	5	5	5	0	0	5	0	5	0
3	5	5	0	5	0	5	0	5	0	5	5	5	0	0	5	0	5	0
31	5	5	0	5	0	5	0	5	0	5	0	0	0	0	5	0	5	0
18	5	5	5	5	5	5	5	5	5	5	5	5	0	0	5	5	5	5
32	5	5	5	5	5	5	5	5	5	5	5	5	0	0	5	5	5	5
9	5	5	5	5	5	5	5	5	5	5	5	5	0	0	5	5	5	5
4	5	5	5	5	5	5	5	5	5	5	5	5	0	0	5	5	5	5
8	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
10	5	5	5	5	5	5	5	5	5	5	0	0	0	0	5	5	5	5
22	5	5	5	5	5	5	5	5	5	5	5	5	0	0	5	5	5	5
16	5	0	0	0	0	0	0	0	0	5	5	5	5	1	0	0	0	0
7	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

Figure 37: Question 5 Scores for Treatment Groups



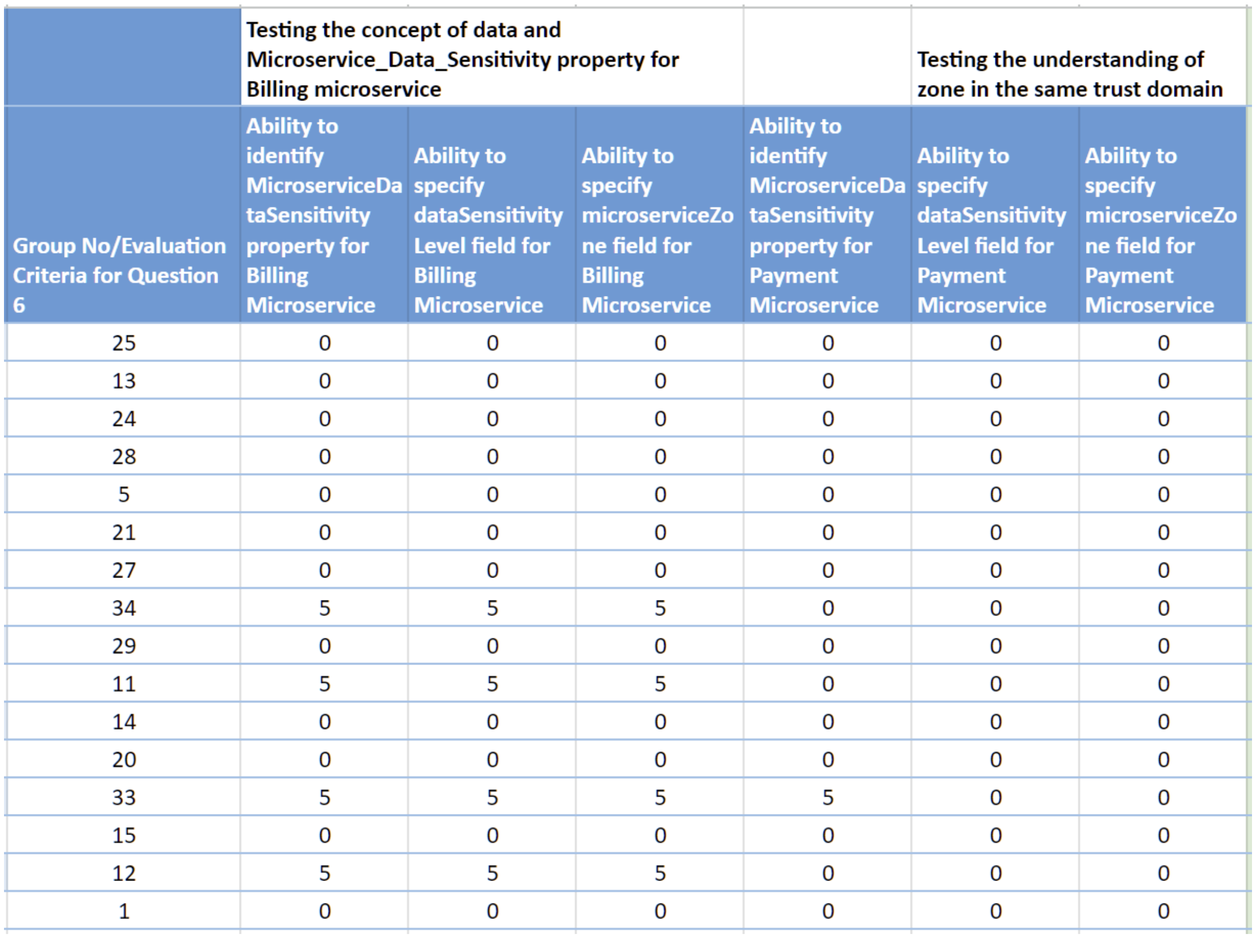
	Testing the structural addition to support the use of EmbeddedPDP as the value for authorizationMicroserviceArchitecture property - PAP									Testing the structural addition to support the use of EmbeddedPDP as the value for authorizationMicroserviceArchitecture property - PIP	Testing the impact of changes of the AuthorizationMicroserviceArchitecture property have on FineGrainedAuthorization property		Testing the impact of changes of the AuthorizationMicroserviceArchitecture property have on CentralizedAccessControl property		Testing the structural addition to support the use of EmbeddedPDP as the value for authorizationMicroserviceArchitecture property - PAP		Testing the structural addition to support the use of EmbeddedPDP as the value for authorizationMicroserviceArchitecture property - PAP	
Group No/Evaluation Criteria for Question 5	Ability to identify the need to add PAP	Ability to identify authorizationMicroserviceArchitecture for PAP	Ability to specify authorizationMicroserviceArchitecture for PAP	Ability to identify fineGrained Authorization for PAP	Ability to specify fineGrainedAuthorization for PAP	Ability to identify centralized AccessControl for PAP	Ability to specify centralized AccessControl for PAP	Ability to identify securityLevelProvided for PAP	Ability to specify securityLevelProvided for PAP	Ability to identify the need to add PIP	Ability to identify fineGrainedAuthorization property needs to be updated fr all core microservice	Ability to specify the fineGrainedAuthorization for all core microservice	Ability to identify CentralizedAccessControl property needs to be updated for all core microservice	Ability to specify the CentralizedAccessControl for all core microservice	Ability to identify dbAccessMode for PAP DB	Ability to specify dbAccessMode for PAP DB	Ability to identify DataAtRestEncryption for PAP DB	Ability to specify DataAtRestEncryption for PAP DB
25	0	0	0	0	0	0	0	0	0	0	5	5	0	0	0	0	0	0
13	5	0	0	5	5	5	5	0	0	5	5	5	5	5	0	0	0	0
24	5	5	5	5	5	5	5	5	5	5	5	3	5	3	0	0	5	5
28	5	5	5	5	5	5	5	5	1	5	5	5	5	5	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	5	5	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	5	5	0	0	0	0	0	0
27	5	5	5	5	5	5	5	5	5	0	5	5	0	0	5	5	5	5
34	5	5	5	5	5	5	0	5	5	5	5	5	0	0	5	0	5	0
29	0	0	0	0	0	0	0	0	0	0	5	5	5	1	0	0	0	0
11	5	5	5	5	5	5	5	5	5	5	5	5	5	5	0	0	0	0
14	5	5	5	5	5	5	5	5	5	5	0	0	0	0	5	0	5	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	5	5	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	5	5	5	1	0	0	0	0
12	5	5	5	5	5	5	1	5	5	5	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	5	5	0	0	0	0

Figure 38: Question 5 Scores for Control Groups



	Testing the concept of data and Microservice_Data_Sensitivity property for Billing microservice				Testing the understanding of zone in the same trust domain	
Group No/Evaluation Criteria for Question 6	Ability to identify MicroserviceDataSensitivity property for Billing Microservice	Ability to specify dataSensitivity Level field for Billing Microservice	Ability to specify microserviceZone field for Billing Microservice	Ability to identify MicroserviceDataSensitivity property for Payment Microservice	Ability to specify dataSensitivity Level field for Payment Microservice	Ability to specify microserviceZone field for Payment Microservice
6	0	0	0	0	0	0
19	0	0	0	0	0	0
2	5	0	0	5	0	0
30	5	0	0	5	0	0
3	0	0	0	0	0	0
31	0	0	0	0	0	0
18	5	5	5	0	0	0
32	0	0	0	0	0	0
9	0	0	0	0	0	0
4	5	0	0	5	0	0
8	5	0	5	5	0	5
10	5	5	5	5	0	0
22	5	5	5	0	0	0
16	5	5	5	5	1	1
7	0	0	0	0	0	0

Figure 39: Question 6 Scores for Treatment Groups

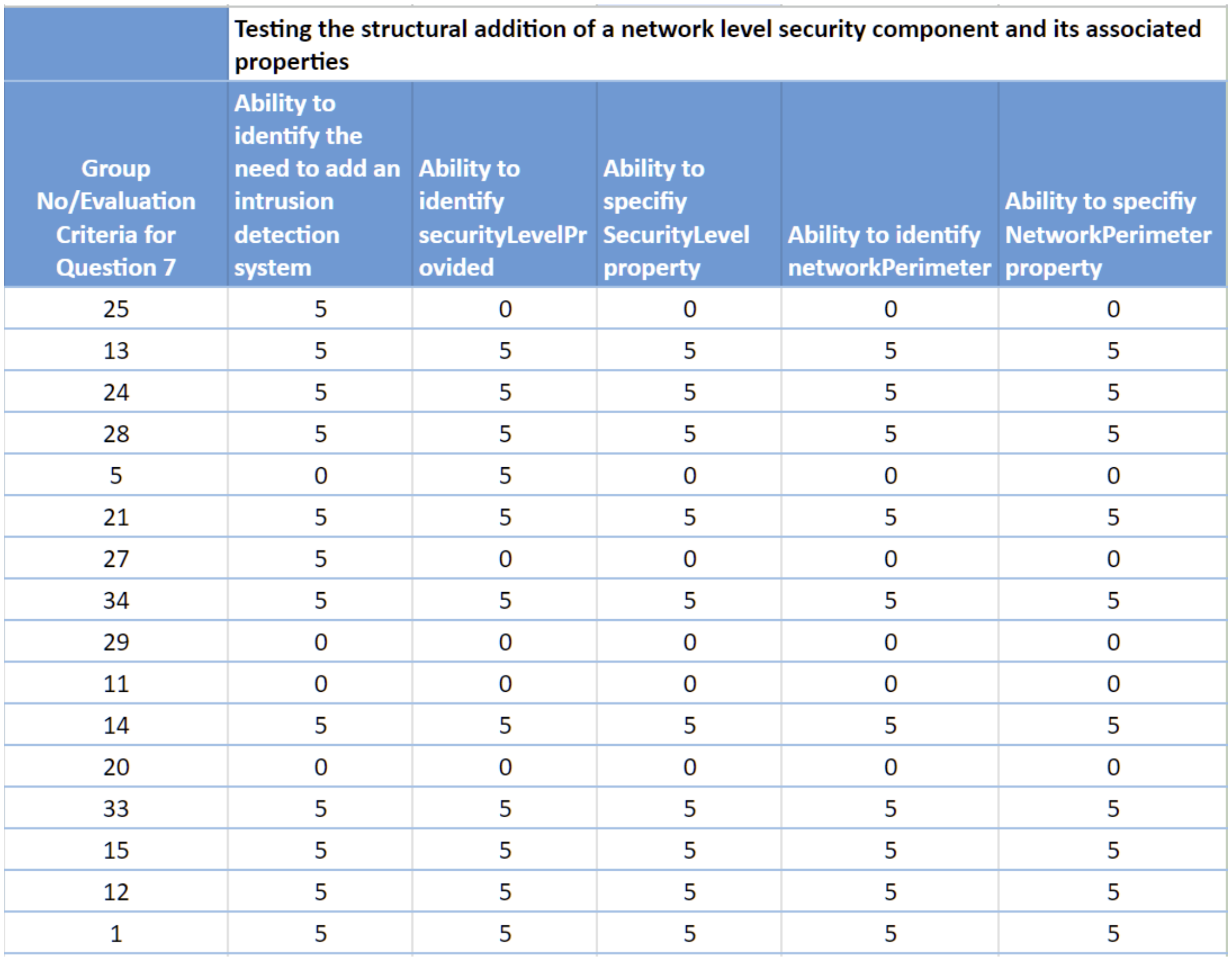


	Testing the concept of data and Microservice_Data_Sensitivity property for Billing microservice				Testing the understanding of zone in the same trust domain	
Group No/Evaluation Criteria for Question 6	Ability to identify MicroserviceDataSensitivity property for Billing Microservice	Ability to specify dataSensitivity Level field for Billing Microservice	Ability to specify microserviceZone field for Billing Microservice	Ability to identify MicroserviceDataSensitivity property for Payment Microservice	Ability to specify dataSensitivity Level field for Payment Microservice	Ability to specify microserviceZone field for Payment Microservice
25	0	0	0	0	0	0
13	0	0	0	0	0	0
24	0	0	0	0	0	0
28	0	0	0	0	0	0
5	0	0	0	0	0	0
21	0	0	0	0	0	0
27	0	0	0	0	0	0
34	5	5	5	0	0	0
29	0	0	0	0	0	0
11	5	5	5	0	0	0
14	0	0	0	0	0	0
20	0	0	0	0	0	0
33	5	5	5	5	0	0
15	0	0	0	0	0	0
12	5	5	5	0	0	0
1	0	0	0	0	0	0

Figure 40: Question 6 Scores for Control Groups

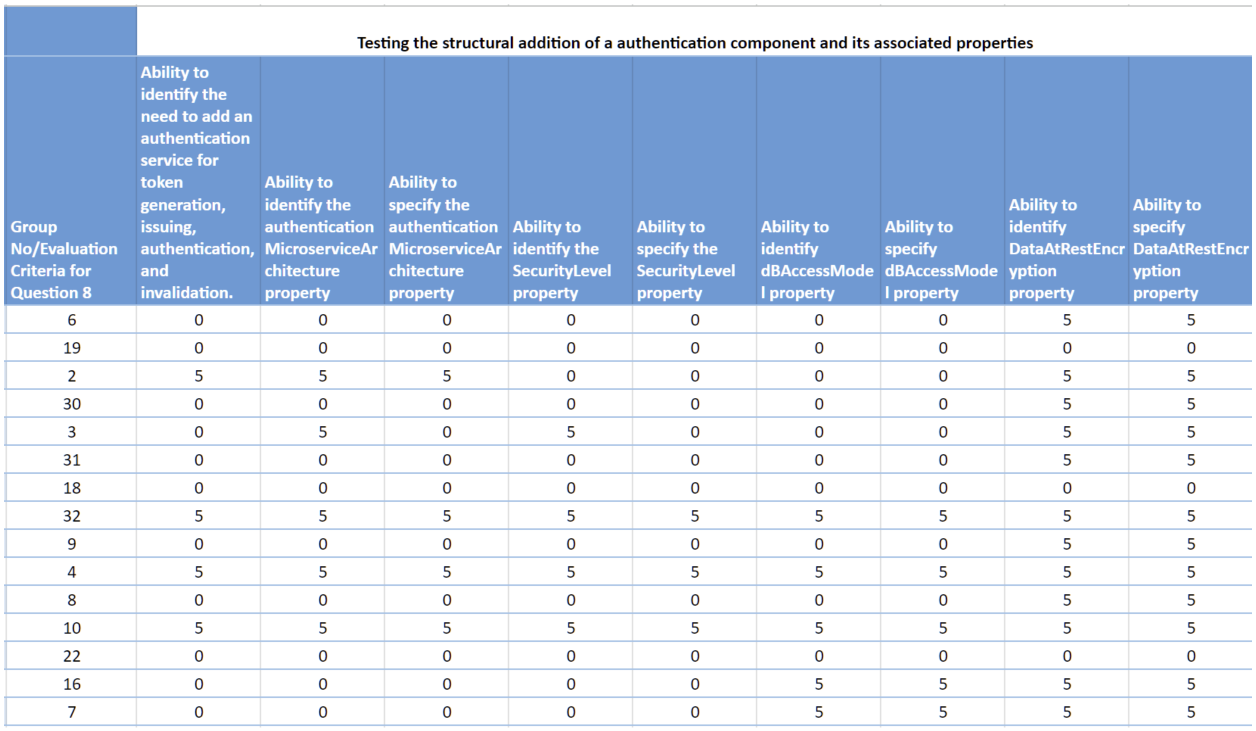
	Testing the structural addition of a network level security component and its associated properties				
Group No/Evaluation Criteria for Question 7	Ability to identify the need to add an intrusion detection system	Ability to identify securityLevelProvided	Ability to specifiy SecurityLevel property	Ability to identify networkPerimeter	Ability to specifiy NetworkPerimeter property
6	5	5	5	5	5
19	0	0	0	0	0
2	5	5	5	5	5
30	5	5	0	5	0
3	5	5	0	5	5
31	0	0	0	0	0
18	0	0	0	0	0
32	5	5	5	5	5
9	5	0	0	0	0
4	5	5	5	5	5
8	5	0	0	0	0
10	5	5	5	5	5
22	5	5	5	5	5
16	5	5	5	5	5
7	5	0	5	0	0

Figure 41: Question 7 Scores for Treatment Groups



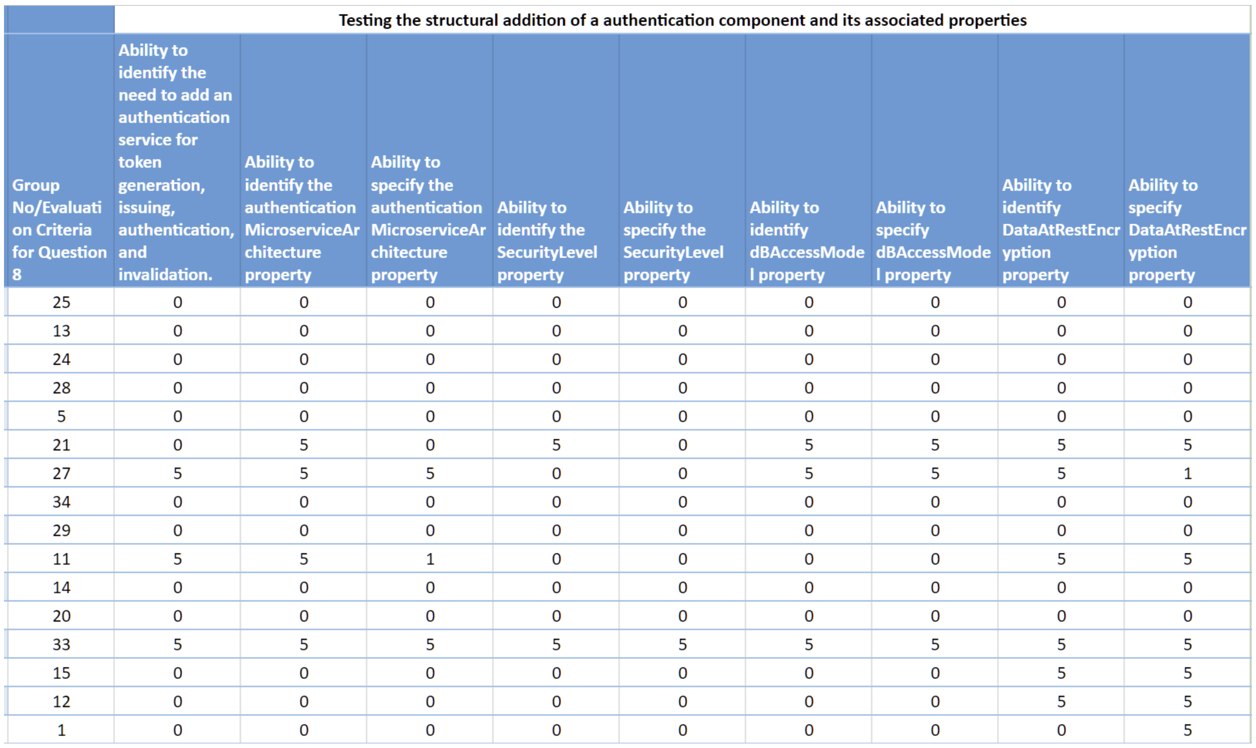
	Testing the structural addition of a network level security component and its associated properties				
Group No/Evaluation Criteria for Question 7	Ability to identify the need to add an intrusion detection system	Ability to identify securityLevelProvided	Ability to specifiy SecurityLevel property	Ability to identify networkPerimeter	Ability to specifiy NetworkPerimeter property
25	5	0	0	0	0
13	5	5	5	5	5
24	5	5	5	5	5
28	5	5	5	5	5
5	0	5	0	0	0
21	5	5	5	5	5
27	5	0	0	0	0
34	5	5	5	5	5
29	0	0	0	0	0
11	0	0	0	0	0
14	5	5	5	5	5
20	0	0	0	0	0
33	5	5	5	5	5
15	5	5	5	5	5
12	5	5	5	5	5
1	5	5	5	5	5

Figure 42: Question 7 Scores for Control Groups



	Testing the structural addition of a authentication component and its associated properties								
Group No/Evaluation Criteria for Question 8	Ability to identify the need to add an authentication service for token generation, issuing, authentication, and invalidation.	Ability to identify the authentication MicroserviceArchitecture property	Ability to specify the authentication MicroserviceArchitecture property	Ability to identify the SecurityLevel property	Ability to specify the SecurityLevel property	Ability to identify dBAccessModel property	Ability to specify dBAccessModel property	Ability to identify DataAtRestEncryption property	Ability to specify DataAtRestEncryption property
6	0	0	0	0	0	0	0	5	5
19	0	0	0	0	0	0	0	0	0
2	5	5	5	0	0	0	0	5	5
30	0	0	0	0	0	0	0	5	5
3	0	5	0	5	0	0	0	5	5
31	0	0	0	0	0	0	0	5	5
18	0	0	0	0	0	0	0	0	0
32	5	5	5	5	5	5	5	5	5
9	0	0	0	0	0	0	0	5	5
4	5	5	5	5	5	5	5	5	5
8	0	0	0	0	0	0	0	5	5
10	5	5	5	5	5	5	5	5	5
22	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	5	5	5	5
7	0	0	0	0	0	5	5	5	5

Figure 43: Question 8 Scores for Treatment Groups



	Testing the structural addition of a authentication component and its associated properties								
Group No/Evaluation Criteria for Question 8	Ability to identify the need to add an authentication service for token generation, issuing, authentication, and invalidation.	Ability to identify the authentication MicroserviceArchitecture property	Ability to specify the authentication MicroserviceArchitecture property	Ability to identify the SecurityLevel property	Ability to specify the SecurityLevel property	Ability to identify dBAccessModel property	Ability to specify dBAccessModel property	Ability to identify DataAtRestEncryption property	Ability to specify DataAtRestEncryption property
25	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
21	0	5	0	5	0	5	5	5	5
27	5	5	5	0	0	5	5	5	1
34	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0
11	5	5	1	0	0	0	0	5	5
14	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0
33	5	5	5	5	5	5	5	5	5
15	0	0	0	0	0	0	0	5	5
12	0	0	0	0	0	0	0	5	5
1	0	0	0	0	0	0	0	0	5

Figure 44: Question 8 Scores for Control Groups

Appendix D: Research Study Background Survey Results

Appendix D presents the result of the research study background survey.

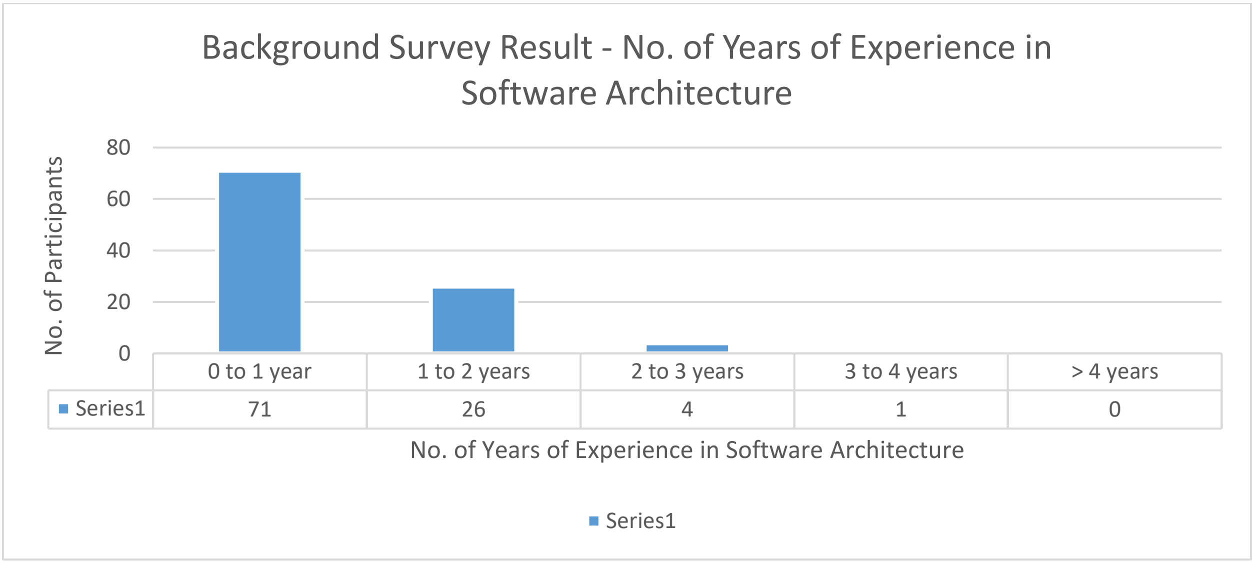


Figure 45: Background Survey Result - No. of Years of Experience in Software Architecture

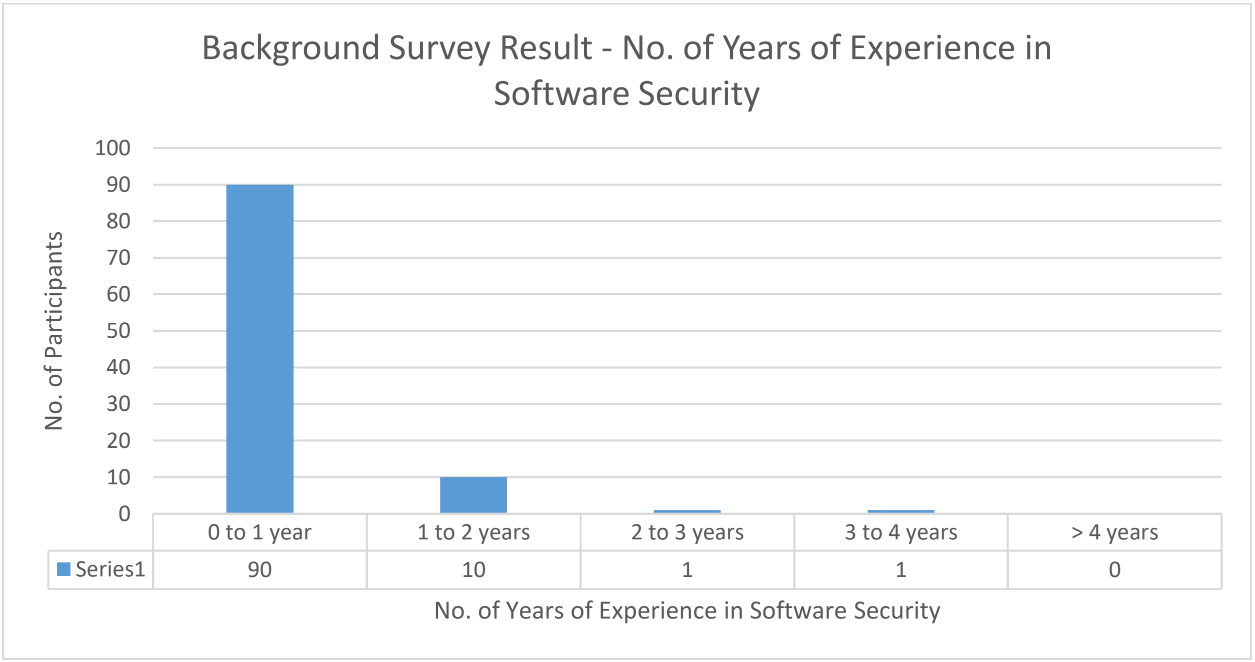
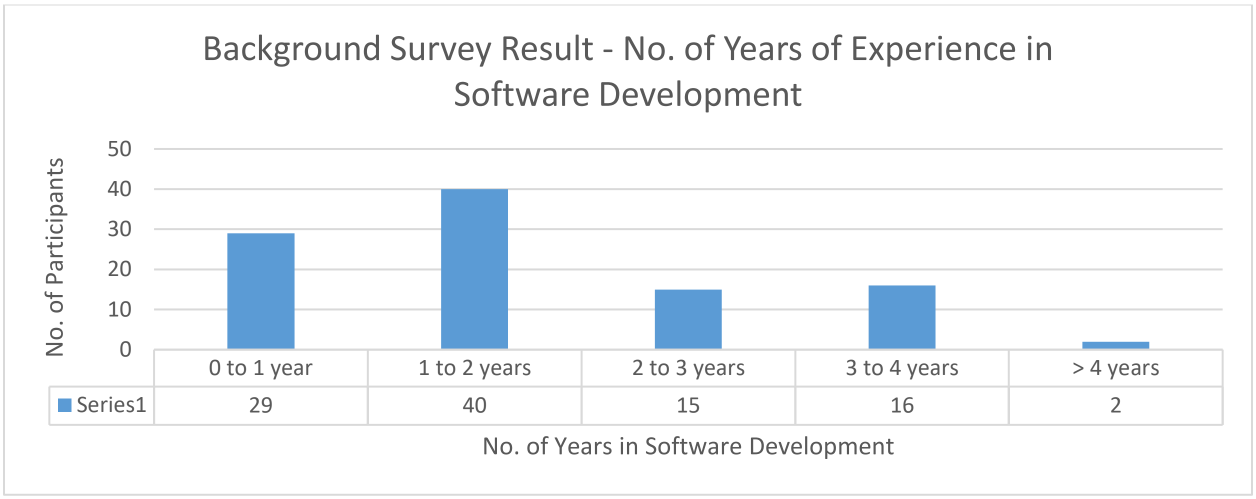


Figure 46: Background Survey Result - No. of Years of Experience in Software Security



	0 to 1 year	1 to 2 years	2 to 3 years	3 to 4 years	> 4 years
Series1	29	40	15	16	2

Figure 47: Background Survey Result - No. of Years of Experience in Software Development

Vita

Wai Yan Elsa Tai Ramirez received her Bachelors of Science (Summa Cum Laude) in Computer Science from The University of Texas at El Paso in the Spring 2004. She received her Master of Science in Computer Science from The University of Texas at El Paso in Spring 2007. In the fall of 2013, she entered the Ph.D. program in Computer Science under the guidance of Dr. Ann Gates. Elsa has 15 years of experience in software requirements engineering and software engineering. Domains she worked in include: audiology, speech-language pathology, digital marketing, telecommunication, cybersecurity, geology, and student success. She completed the Certification in Effective College Instruction by The Association of College and University Educators and The American Council on Education and South East Asia Patent Drafting Participation and Completion Certificate from Fédération Internationale des Conseil en Propriété Industrielle (FICPI) Academy of Education.

While pursuing her Ph.D. degree, she worked as a lecturer and taught the following courses: Software Engineering I: Requirements Engineering (Part 1 of the Software Engineering capstone course undergraduate level), Advanced Object-Oriented Programming (undergraduate level) Software Requirements Engineering (graduate level), and Software Architecture and Design (graduate level).

During her time at UTEP, she received the CAHSI-Google Dissertation Award.

Contact Information: wyetai@utep.edu

This dissertation was typed by Wai Yan Elsa Tai Ramirez.