MICROUSITY : SECURITY TESTING TOOL FOR BACKEND FOR FRONTEND (BFF) MICROSERVICES

ไมโครยูซิตี้ เครื่องมือทดสอบความปลอดภัยสำหรับไมโครเซอร์วิส แบบแบคเอ็นด์ฟอร์ฟร้อนต์เอนด์ (บีเอฟเอฟ)

BY

MISS. CHANSIDA	**MAKARANOND**	**6188010**
MR. PATTARAKRIT	**RATTANUKUL**	**6188018**
MR. PUMIPAT	**WATANAKULCHARUS**	**6188026**

ADVISOR
DR. CHAIYONG RAGKHITWETSAGUL

CO-ADVISOR
DR. VASAKA VISOOTTIVISETH

A Senior Project Submitted in Partial Fullfillment of the Requirement for

THE DEGREE OF BACHELOR OF SCIENCE (INFORMATION AND COMMUNICATION TECHNOLOGY)

Faculty of Information and Communication Technology
Mahidol University

2021

ACKNOWLEDGEMENTS

This endeavor would not have been possible without the assistance of several individuals. Dr. Chaiyong Ragkhitwetsagul has been an amazing instructor, mentor, and thesis supervisor, providing sound guidance and support with just the right amount of insight and levity. We are proud of and appreciative of the time I spent working with him. We are incredibly appreciative that you accepted us as supervisees and maintained trust in us throughout the years. Additionally, Assoc. Professor Dr. Vasaka Visoottiviseth, who serves as our co-advisor and provides direction and assistance to our team. We have profited tremendously from your depth of expertise and painstaking editing.

Additionally, thanks to Mr. Tanapol Nearunchorn and Mr. Chanon Khamronyutha from Lineman Wongnai (LMWN) compnay for your extraordinary generosity and for accepting us as part of your LMWN intern team. Additionally, for assisting us along the process, offering advice and direction, and providing us with access to LMWN resources for testing and study.

Finally, we would like to express our gratitude to the countless people that supported and loved us during this lengthy journey. Thank you to our loved ones for constantly listening to our complaints and talking things out, for proofreading repeatedly even after long days at work, from different time zones. Also, thank you for believing in me, supporting me and for making sacrifices to help me complete this project. We will be eternally grateful for the unwavering love and support we received during the thesis process and on a daily basis.

Miss. Chansida Makaranond

Mr. Pattarakrit Rattanukul

Mr. Pumipat Watanakulcharus

MICROUSITY : SECURITY TESTING TOOL FOR BACKEND FOR FRONTEND (BFF) MICROSERVICES

MISS. CHANSIDA MAKARANOND 6188010 ITCS/B
MR. PATTARAKRIT RATTANUKUL 6188018 ITCS/B
MR. PUMIPAT WATANAKULCHARUS 6188026 ITCS/B

B.Sc.(INFORMATION AND COMMUNICATION TECHNOLOGY)

PROJECT ADVISOR: DR. CHAIYONG RAGKHITWETSAGUL

ABSTRACT

Today, microservice architecture is widely used in corporate software development. This is because microservices-based software development enables software development to be more scalable and more efficient than traditional processes. The creation of microservices software requires the extensive usage of Application Programming Interfaces (APIs). However, vulnerability scanning and API functional testing are not catching up. Thus, vulnerabilities such as data disclosure or excessive data exposure might arise. Also, some of the currently API security testing tools do not provide report visualization, which makes it extremely difficult for the developers to comprehend the information and fix the issues. Finally, it is important to provide self-assisted learning platform for interested persons about understanding and avoiding API attacks.

We aim to solve the aforementioned problems. Microusity is an application that performs API security testing on a popular type of microservice called Backend for Frontend (BFF) systems. With the adoption of state-of-the-art fuzzing technique and network monitoring plus the ability to create visualizations, Microusity will help security testers to detect API security issues and comprehend the detected issues via the tool's graph visualization report and education content related to the detected issues. The program also has a sandbox simulation system that allows learners to learn and encourages API security learners to test and practice on their own.

KEYWORDS: API SECURITY, MICROSERVICES, FUZZING 88 P.

ไมโครยูซิตี้ เครื่องมือทดสอบความปลอดภัยสำหรับไมโครเซอร์วิส
แบบแบคเอ็นด์ฟอร์ฟร้อนต์เอนด์ (บีเอฟเอฟ)

นางสาว จันสิดา มกรานนท์ 6188010 ITCS/B
นาย ภัทรกฤต รัตตานุกูล 6188018 ITCS/B
นาย ภูมิภัทร วัฒนกุลจรัส 6188026 ITCS/B

วท.บ. (เทคโนโลยีสารสนเทศและการสื่อสาร)

อาจารย์ที่ปรึกษาโครงการ: ดร. ชัยยงค์ รักขิตเวชสกุล

บทคัดย่อ

ปัจจุบัน สถาปัตยกรรมแบบไมโครเซอร์วิสมีการใช้กันอย่างแพร่หลายในการพัฒนาซอฟต์แวร์ขององค์กร เนื่องจากการพัฒนาซอฟต์แวร์บนไมโครเซอร์วิสมีศักยภาพที่จะทำให้การพัฒนาซอฟต์แวร์ตรงไปตรงมาและมีประสิทธิภาพมากกว่ากระบวนการแบบเดิม การสร้างซอฟต์แวร์ไมโครเซอร์วิสต้องใช้ Application Programming Interface (API) อย่างไรก็ตาม การค้นหาช่องโหว่และการทดสอบการทำงานของ API ไม่ได้ถูกใช้อย่างกว้างขวาง ซึ่งหมายความว่าอาจเกิดช่องโหว่ เช่น การเปิดเผยข้อมูลหรือการเปิดเผยข้อมูลมากเกินไป นอกจากนี้ ยังไม่มีการทดสอบการทำงานของ API อย่างกว้างขวาง ดังนั้นอาจเกิดช่องโหว่เช่นการเปิดเผยข้อมูลที่มากเกินไป นอกจากนี้ เครื่องมือที่มีอยู่ในปัจจุบันยังมีข้อจำกัดและไม่ได้ให้การแสดงภาพรายงานแบบกราฟ ทำให้นักพัฒนาสามารถเข้าใจข้อมูลเกี่ยวกับช่องโหว่ภายในซอฟต์แวร์ได้ยากมาก นอกจากนี้ ความรู้และความตระหนักด้านความปลอดภัยของ API ยังไม่มีแหล่งรวบรวมที่สามารถเข้าถึงได้ง่าย ด้วยเหตุนี้ จึงมีความเหมาะสมที่จะมีคำแนะนำด้านการศึกษาและความรู้ซึ่งเป็นส่วนหนึ่งของการให้ความรู้แก่นักศึกษา คณาจารย์ และผู้สนใจเกี่ยวกับการป้องกันการโจมตีและการรักษาความปลอดภัยสำหรับ API ในส่วนเกี่ยวกับไมโครเซอร์วิส ดังนั้นทีมวิจัยจึงคาดคะเนปัญหาและเล็งเห็นปัญหาและมุ่งแก้ปัญหาเหล่านั้น

ระบบ Microusity เป็นเว็ปแอพพลิเคชั่นที่เสนอเพื่อแก้ปัญหา API 3:2019: Excessive Data Exposure ภายในรูปแบบสถาปัตยกรรมแบบไมโครเซอร์วิสดังนั้น Microusity เป็นเว็ปแอปพลิเคชันที่ทำการทดสอบความปลอดภัย API บนระบบ BFF ด้วยความสามารถในการสร้างภาพข้อมูลเพื่อช่วยให้ผู้ทดสอบความปลอดภัยเข้าใจปัญหาด้านความปลอดภัยได้อย่างง่ายดายผ่านการใช้รายงานการสร้างภาพกราฟ Microusity โปรแกรมยังมีระบบจำลองแซนด์บ็อกซ์เพื่อส่งเสริมให้นักเรียนได้ลองทดสอบและฝึกฝนด้วยตนเอง

88 หน้า

CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1
INTRODUCTION

1.1 Motivation

When the software architecture is being released into production, it is built with flexibility in mind. Hence, the work of system architecture has no lifespan, and it will continue to be used until there is a better implementation. Throughout time, the evolution of software architecture has played an essential role in our software development and implementation process. In the past, monolithic software architecture was considered one of the commonly used software architectures. A monolithic software architecture is built as a single logical executable. Scaling monolithic programs usually provides a challenging task for the developer because every modification impacts the entire system since monolithic works as a single unit [17]. Also, [18] if one service fails, everything stops operating in a monolithic service. In order to overcome the restrictions of monolithic architecture, microservices were created to address these problems.

Microservices are small autonomous services that function with lightweight coding mechanisms. The microservices architecture enhances the traditional monolithic architecture by utilizing technologies to separate the application processes to establish an autonomous small service group operating in its own processes. Many enterprises have found that by employing sophisticated microservices architectures, they can build software more quickly and leverage new technologies. Compared to the traditional monolithic services [17], microservices architecture provides many benefits. For instance, microservices enable technology heterogeneity, resilience, scalability, ease of deployment, organizational alignment, composability, and more [17]. Furthermore, microservices also give the advantages of making decisions. Therefore, more alternatives and more selections will be available from the use of microservices architectures. Because decisions are considerably more prominent in this microservices framework than in simpler, monolithic systems [17]. One of the most prominent characteristics of a microservice

is its autonomous functionality. Their deployment independence best characterizes microservices. As a result, a system must not be affected if any service is interrupted, shut down, or changed [19]. In addition, each microservice has its own database to store information. With this, the developer may easily regulate the database's performance and scalability. Besides, if the developer wishes to access a microservice's database, the change must be called only through that service's API [18].

1.1.1 The Rise of Microservices

Microservices are one of the most popular terms today [17] and the use of microservices has been prominently increasing nowadays. Online services are increasingly moving toward a microservice architecture. For instance, the world's leading company, Netflix, has shifted from a traditional architecture to a microservice architecture, using the Backend For Frontend (BFF) design pattern to solve the complex system management problems it encounters. From the traditional architecture system, microservices architecture makes it easier for Netflix companies to plan for managing systems and huge amounts of data [20]. As a result, strategic planning and implementation efforts at numerous prominent organizations have given microservices close attention and importance. According to research firm Research and Markets, microservices market growth will be driven by the benefits of microservices architecture and hybrid clouds across various end-user sectors [21]. Furthermore, according to firm researchers and market experts, the microservices industry will grow at 22.5 percent annual rate over the next five years [21].

1.1.2 The Rise of APIs Security

Every day, we all utilize hundreds of APIs in today's data-driven economy to access and exchange data. Most sensitive data is kept in databases, such as financial, healthcare, and personally identifiable information (PII), which may possibly be routinely stored in an unsecured and inaccurate manner. As a result, theft or manipulation of such data is possible. In particular, API security concerns can occur in microservices systems that lack appropriate security mechanisms. In accordance with this, SALT Security company's investigation report on the current state of API security highlights that

API traffic accounts for more than 80 percent of total Internet traffic [22]. Moreover, according to a recent assessment of the literature in this field, the usage of APIs and the value of data transmitted over them has proliferated. Thus, the statistics reveal that from 195 million API calls in December 2020, the number of API calls increased to 470 million API calls in June 2021 [23].

As reported in a recent literature review on this subject, attackers' interest has risen as API usage and the value of data transferred has expanded [22]. Also, several studies, for example [22], [23], and [24] have revealed that APIs are often the weakest link in a company's application security chain, and attackers are aware of this. Despite that, many companies still fail to maintain the security of their API for many reasons, such as lack of awareness, overconfidence in traditional security tools, and more [25].

Remarkably, the primary cause of API security difficulties originates from too much reliance on developers to handle API security issues. Based on the study of the current state of API 2021, 36 percent of respondents feel that developers or DevOps teams are primarily responsible for safeguarding APIs, according to the study's results. It is not only impractical, but it puts a company at risk [23].

1.1.3 Visualization for System Security

Furthermore, without a proper visualization tool to display the monitoring data, the data from the security detection tool can become burdensome for the user. As stated by Yang Cai, Director of the CyLab Visual Intelligence Studio at Carnegie Mellon University, "It's quite difficult for analysts to look through 80 columns in a table" [25]. Moreover, Cai believes that with the benefit of gathering thousands of network data and representing them in an easy pattern using the visualization tools, this can help a detector of anomalies attack much faster than the traditional process [25]. Thus, it is beneficial to implement sufficient visualization tools to display the overall result of the security detection tools. With the visualization tool, it can significantly reduce the complexity of multiple data sets. Therefore, complex data can be identified and graphically visualized using a visualization tool such as the data dashboard. As a result, the developer can discover the risk, analyze risks, observe the vulnerable spots, and respond faster to identifying the breach points in the system.

From the literature, when dealing with sensitive data, the software must take additional precautions. API security should be a key priority, especially for companies that rely on application development to promote their business innovation. Consequently, detecting problems before a security breach occurs can be very beneficial to the system's productivity and performance.

1.2 Problem Statements

This project tackles the following problems as follow:

1. The existing tools for evaluating BFF systems for sensitive data exposure identify just the errors, but cannot precisely identify the source of the errors when one-to-many requests are created by BFF.
2. The results from API security testing tools can be difficult to comprehend without a proper visualization display.
3. Lack of relevant training and education resources in Thailand's computer courses on API security problems are also few and fall behind the expanding trend of harmful user software assaults.

1.3 Objectives of the Project

The objectives of the project are as follows:

1. To create an automated API security testing tool for BFF systems.
2. To implement a visualization that easily displays and views the results of API security testings.
3. To integrate the result of API security testing into continuous integration as part of the software development pipeline.
4. To educate and raise the learner's awareness about attack avoidance and microservices API security.

1.4 Scope of the Project

The project falls under the following scope:

1. The proposed automated API security testing tool is available as a web application.
2. The proposed automated API security testing tool only supports BFF design pattern for microservices architecture style.

1.5 Expected Benefits

This project provides the following expected benefits:

1. The proposed automated API security testing tool for checking the API security of microservices systems can be used by software companies to improve their systems' API security.
2. The software companies are more aware of potential vulnerable API security risks in their microservices system.
3. The software development is strengthened and more guarded from API security risks during development time.
4. The need of attack prevention and API security is now being recognized. As a result, the software's security is effectively safeguarded.

1.6 Organization of the document

The document consists of 6 parts including Introduction (Chapter 1), Background (Chapter 2), Analysis and Design (Chapter 3), Implementation (Chapter 4), Evaluation Results (Chapter 5), and Conclusion (Chapter 6). Firstly, The Introduction chapter includes motivation, problem statements, objectives, scope, expected benefits, and organization of the document. Secondly, the Background chapter describes the overview of the project, which has fundamentals and related work. Thirdly, the Analysis and design chapter contains work procedures: methodology, system architecture, structure chart, and system analysis.

CHAPTER 2
BACKGROUND

The background and information necessary for completing this project is presented in this chapter. The first component is the section Fundamentals, which contains fundamental project information. The second part is Tools and Technologies, which covers the tools and techniques used for the project. The third component is a section on a literature review which provides an overview of project-related research papers.

2.1 Fundamentals

This section is to provide the basic knowledge of the project fundamental terms that necessary for understanding this project.

2.1.1 Web API or Web Service

In our modern technology-driven era, web API and web service are the two most common terms. Despite their ability to operate together, web APIs and web services are not the same things. The differences can be explained as follows. First, web APIs are based on the concept of API. An API is a programming interface that enables communication between different applications [26]. The Web API can be implemented on a web server or in a web browser. When client devices such as smartphones, laptops, and desktops make queries to the web API, the web API will forward those requests to the webserver for processing and then deliver the processed result to the client. Web services are used to connect with the cloud and exchange data. Nowadays, it is essential to be able to share data across various devices and apps instead of being kept on the device itself. A web service is a general word that refers to an interoperable machine-to-machine software function hosted on a network-accessible server [27]. Nowadays, having a web service is critical in today's tech-driven society since web services facilitate the development of software architectures. In general, there are several types of web service protocols or standards, such as XML-RPC, UDDI, SOAP, REST, and JSON. According to

IBM CICS®, web service protocols like SOAP and JavaScript Object Notation (JSON) are two of the most distinguishable types of web service [27]. It is important to note that each of these methods has its advantages and disadvantages. Also, there is a primary distinction between web service and web API.

On the one hand, SOAP is a standardized protocol that facilitates the transmission of messages across other protocols such as HTTP and SMTP. The World Wide Web Consortium maintains and develops the SOAP specifications as official web standards (W3C) [28]. SOAP is an XML-based communications system for sending and receiving data over the internet between two computers and web services. SOAP works by allowing the client to encapsulate a method call in SOAP/XML, subsequently sent to the server over HTTP. The XML request is processed to determine the method name and arguments provided, and processing is delegated. As a result, a SOAP message's envelope specifies the message's start and endpoints. In addition, since SOAP messages are entirely composed of XML, they may be used on any platform and in any language.

On the other hand, most programming languages allow the developer utilizes JavaScript Object Notation (JSON). JSON data format because JSONdata is composed of collections of name/value pairs and ordered lists of values. Both of these are common data structures utilized by the majority of computer languages. It is based on the Javascript/ECMAScript scripting language. JSON is now the de facto industry standard for data exchange between web services, web applications, browsers, and mobile apps [29]. JSON can offer many benefits, such as its straightforward form, modifying flexibility and adaptability, making it simple to read and comprehend. Contrary to common perception, JSON does not need only the JavaScript programming language, but it can be adapted to many programming languages.

With the proliferation of AJAX-powered websites, it is becoming more essential for websites to load data fast, asynchronously, or in the background without slowing page display. A large number of existing studies in the literature [30], [31], [32] had examined the usages of different web service data formats that may affect the performance of an application. The literature review from the Department of Computer Science and the Information Technology, University of Oradea, Romania, suggested that JSON is more proficient compared to SOAP. Due to the fact that JSON facilitates the transmission of

data across various devices and applications to a greater extent [33]. Moreover, based on the in-depth analysis of SOAP and JSON performance with web API in the PHP framework, study [30] reveals that both in terms of data size and response time for web APIs, JSON format offers 30-40 percents quicker transmission than SOAP XML format. For XML Web Application Programming Interface, the JSON data format outperforms the others in terms of response speed and data volume (API). However, SOAP still provides superior support for specific applications that need the transmission of several heterogeneous XML formats [31]. Also, the JSON data format uses less memory on average than the XML data format [32].

In order to interact with hardware components, such as media components, files, or objects, the RESTFUL library was developed for this purpose. When it comes to web services, those that conform to the REST model are referred to as RESTful. Standard HTTP verbs like GET, POST, PUT, and DELETE is used to interact with the required components through REST.

Therefore, based on the data gathered from the literature review, our project has chosen the JSON framework as the main data exchange format. Because of its benefits, SOAP is governed by rigorous standards and has sophisticated security measures since it is an established protocol. Therefore, it makes SOAP require more complexity and necessitates additional bandwidth and resources, resulting in slower page load times. Thus, JSON data is more suitable and may be completed through web APIs in a timely manner, with fewer delays to the end-users.

2.1.2 Modern API-based Service Development

Although web APIs have existed for over two decades, it is only in the last few years that the term “web API.” has received considerable attention. As the number of developers adopting an API-centric approach to developing software and online applications continues to grow rapidly, Remarkably, API development philosophy has changed over time to meet the contemporary requirements of business architecture, and developers must understand how to adapt to these current API design concepts.

2.1.3 Monolithic Software Architecture

Monolithic applications dominated the industry for decades. Monolithic software architecture refers to a structure that is entirely made of one component. Thus, a monolithic application is a single-tiered software application in which several components are integrated into a single program running on a single platform [34]. Many monolithic web apps make use of JSON through REST APIs. When deploying a program, it may be a single file (like Java) or a group of files residing in the same directory.

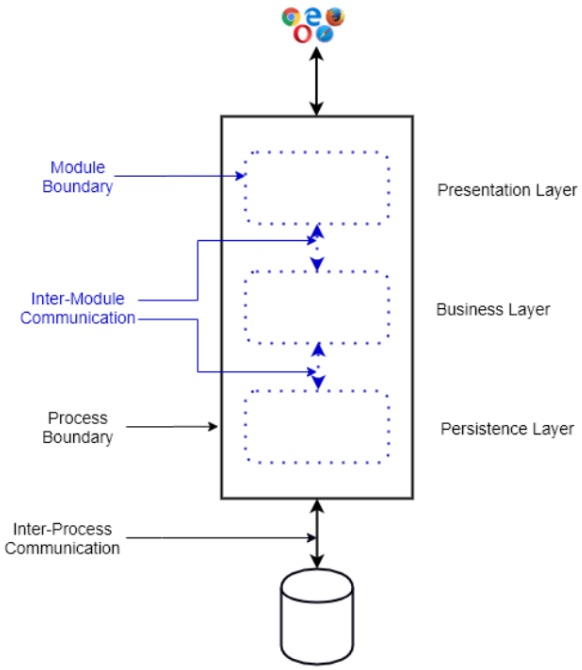


Figure 2.1: Monolithic Web Application [1]

According to the diagram in Figure 2.1, the whole application is hosted on a web server and application server. The application is split into several levels yet remains intact as a whole (Presentation, Business, and Persistence). Undoubtedly, the most distinct characteristic of monolithic architecture is its deployment as a whole.The application is executed in a single process. Also, based on the illustration, there is just one online transaction processing database in a monolithic application. As a consequence, managing transactions and exchanging data is simple.

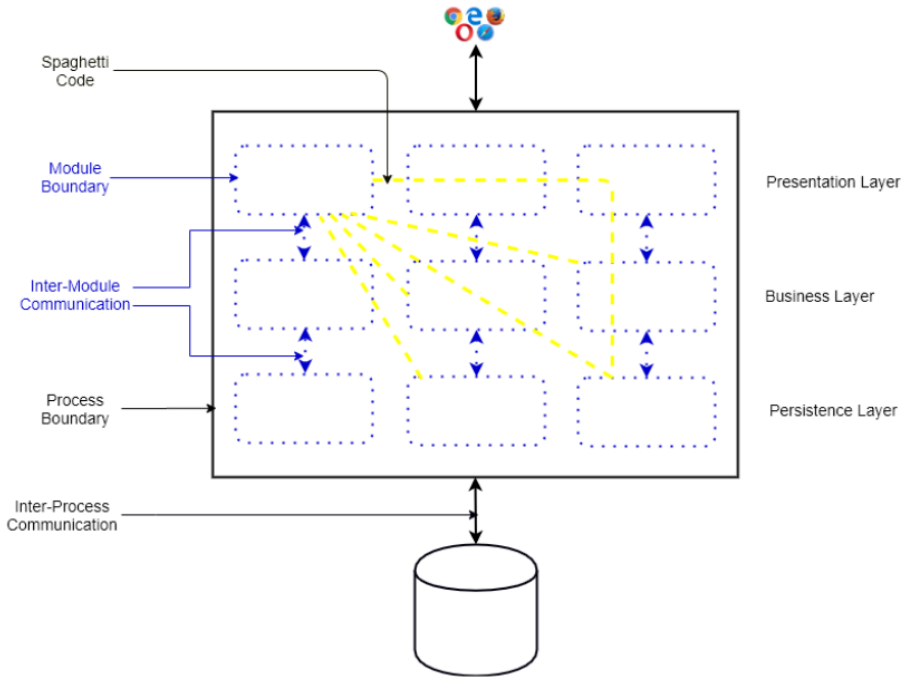


Figure 2.2: Large Monolithic Web Application [1]

While monolithic systems are initially simple to operate, their size and complexity grow with time. The drawbacks of monolithic applications become considerably more important as the application grows in size and user base as in Figure 2.2. There are several disadvantages to monolithic applications, such as scalability problems. Firstly, monolithic apps were inefficient at scaling, especially because of the shared code-base, when different components had different resource requirements. Due to that fact, it can only be scaled on a one-dimensional scale. Thus, scaling a large monolithic application has become a time-consuming task for developers. Because a developer needs to completely reinstall the software in order to modify a single component [35]. As a result, parallelizing tasks across different teams is challenging. Secondly, with such a big and sophisticated application, making changes is more difficult due to the tight coupling. Any modification to the source code has ramifications across the whole system, therefore planning is essential. Hence, the entire development process would be prolonged as a result of this issue. Additionally, monolithic structures impose obstacles to the adoption of new technologies. The implementation of a new technology in a monolithic program is very difficult since the whole application must be rebuilt[36].

The Figure 2.2 represents a large monolithic web application system structure.

Based on the case, there is an obvious problem with the system when the business wants to scale up its components within the layers.

To sum up, monolithic software architecture evolved together with the development of software systems, as new methods were created to meet the specific challenges of each era. In today's world, architecture has become an ingrained element of contemporary activity. As a result, the software architectural style seems to be a main force that drives our business and our world, therefore, software architecture should be emphasized.

2.1.4 Microservice Software Architecture

DevOps (A compound of development (Dev) and operations (Ops)), cloud computing infrastructure, mobile computing, and elastic computing have all contributed to the rise of the microservice software architectural, which now competes with the monolithic approach to building extensive software systems. In order to tackle this problem, the microservices software architecture has been developed.

In the past, only individuals with PhDs in science and mathematics could utilize the first programming languages because of the high entry barriers that existed. As mentioned by Rockmore about the ability to access the computer machine in the 1960s , Rockmore quoted that "Behind locked doors, only guys–and, once in a while, a woman–in white coats were able to access the computer machine." As time passed by, non-PhD students from various fields may have participated in the development of software, causing computer applications to expand rapidly in the 1960s [2]. The BASIC programming language was created in 1964 as a general-purpose programming language [2]. Therefore, the 1960s marked a revolutionary change for software development. The software has grown in size and complexity. The old traditional principle of monolithic architecture design was removed, and the new strategy of "Divide and Conquer" was established and used by computer scientists in an attempt to conquer the complexity of software systems.

Figure 2.3: John Kemeny (left) and Thomas Kurtz (center) discuss a program with a Dartmouth student early in BASIC's existence [2]

DevOps (A compound of development (Dev) and operations (Ops)), cloud computing infrastructure, mobile computing, and elastic computing have all contributed to the rise of the microservices architecture style, which now competes with the monolithic approach to building extensive software systems. In order to tackle this problem, the microservices software architecture has been developed.

Microservices is a methodology for building a single application as a collection of smaller services, each of which runs in its own process and communicates via lightweight methods, most often an HTTP resource API. With microservices, big software projects may be divided into smaller ones with loosely connected modules that interact with one another through simple APIs. Microservices have lots of advantages over traditional monolithic software architectures.

Firstly, one of the main benefits of microservices is their application scalability features. Because each microservices may operate independently, adding, removing, updating, and scaling individual microservices is very simple. Therefore, rather than expanding a whole application, developers may provide more resources to the most critical microservices. This also implies that scaling is more rapid and often more cost-effective with a microservices architecture as in Figure 2.4 and Figure 2.5.

Secondly, microservices enhance quick development. With microservices, new products and features can be created and released more quickly because each unique feature may be produced as a standalone microservices that can be tested and deployed

separately and quickly. Additionally, microservices aid the DevOps mentality. Due to companies' fast development and deployment of new products, customers see the results of their input in weeks, not months or years. This fosters cooperation between users, developers, and engineers, thus improving user retention and happiness.

Thirdly, fault isolation in Microservices reduces downtime. Microservices enable developers to isolate errors to a single service, avoiding cascading failures that would otherwise cause the program to crash. It is possible to avoid cascading failures by using features like "Circuit breakers" in microservices design, which prevent server resource depletion when a calling service is forced to wait on hold for a non-responding service that has failed [37]. Because of this fault isolation, even if one of the application's modules fails, it will continue to function normally.

Fourthly, the security aspect is one of the key benefits of microservices. With microservices, the security of data becomes the main priority and is taken into consideration when developers create connections across microservices. Therefore, a secure API guarantees that the data in the microservices processes is accessible only to approved apps, users, and servers. Accordingly, when it comes to sensitive health, financial, or other kinds of private data, microservices enhance a secure API, which allows developers to maintain complete control over which data is accessible to the more extensive application and its users. Making it possible to comply with HIPAA [38], GDPR [39], and other data security laws.

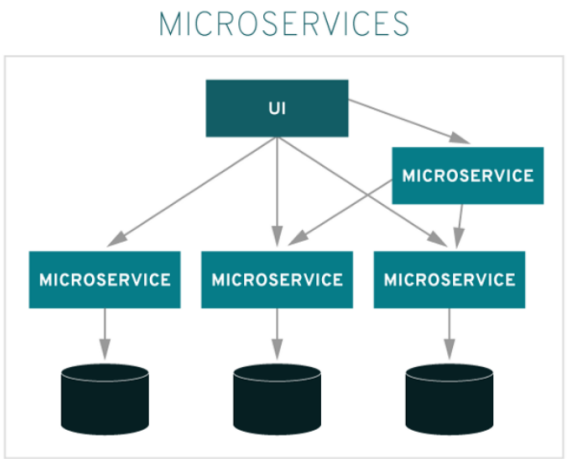


Figure 2.4: Microservices architecture diagram [3]

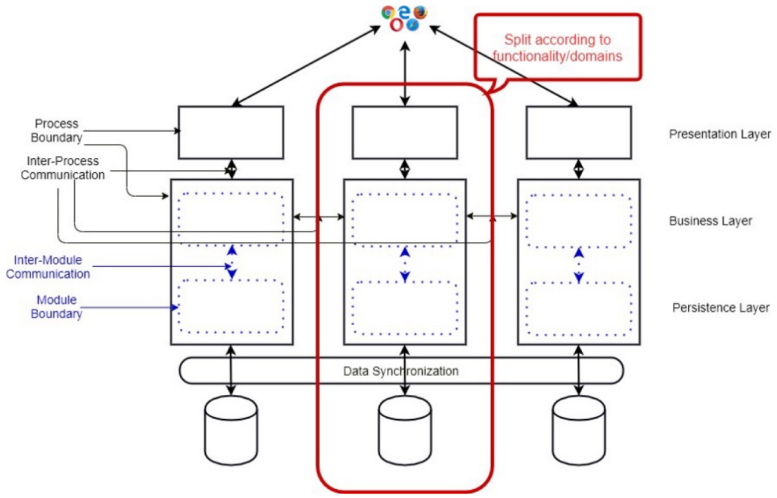


Figure 2.5: Microservices architecture structure split according to functionality domain [1]

2.1.5 Microservice Implementation

When implementing the microservice architecture design pattern, a typical microservices architecture (MSA) should have the following components as in Figure 2.6 [4]. Each of the components can be explained as follows.

Figure 2.6: Core component of microservices [4]

Client

One of the earliest components of microservices is clients. The architecture starts with clients from various devices attempting to execute administration functions. For instance, searching, building, configuring, etc. Furthermore, if consumption is carried out directly, the client must deal with many requests to microservice endpoints.

Identity Providers

When a customer transmits a request, client queries are routed to identity providers, who authenticate the customers' demands and transmit them to API Gateway. The requests are then routed via a well-defined API Gateway to the internal services. Significantly, user-driven and server-to-server access to identity data must be supported by the Identity Microservices.

API Gateway

A microservices architecture must also have an API gateway. Because they function as the primary abstraction layer between microservices and external clients, API gateways are essential for communication in a distributed architecture.

In order to make requests to the appropriate microservices, clients must go via API Gateway, which acts as a point of entry for them. Thus, a microservices-to-client API gateway standardizes the translation of messaging protocols, which frees both the service provider and client from having to translate requests in unknown forms. Additionally, most API gateways provide security capabilities such as permission and authentication management for microservices, along with the ability to monitor incoming and outgoing requests in order to detect any unauthorized access.

Messaging Formats

Messages of the Synchronized and Asynchronized classes are used to communicate between them [40].

Asynchronous message: According to the Figure 2.7, communication between microservices must be supported by every microservice. The usage of AMQP, STOMP, and MQTT protocols is common in situations where customers don't have to wait for service replies. Due to the nature of messages being specified, these protocols are utilized

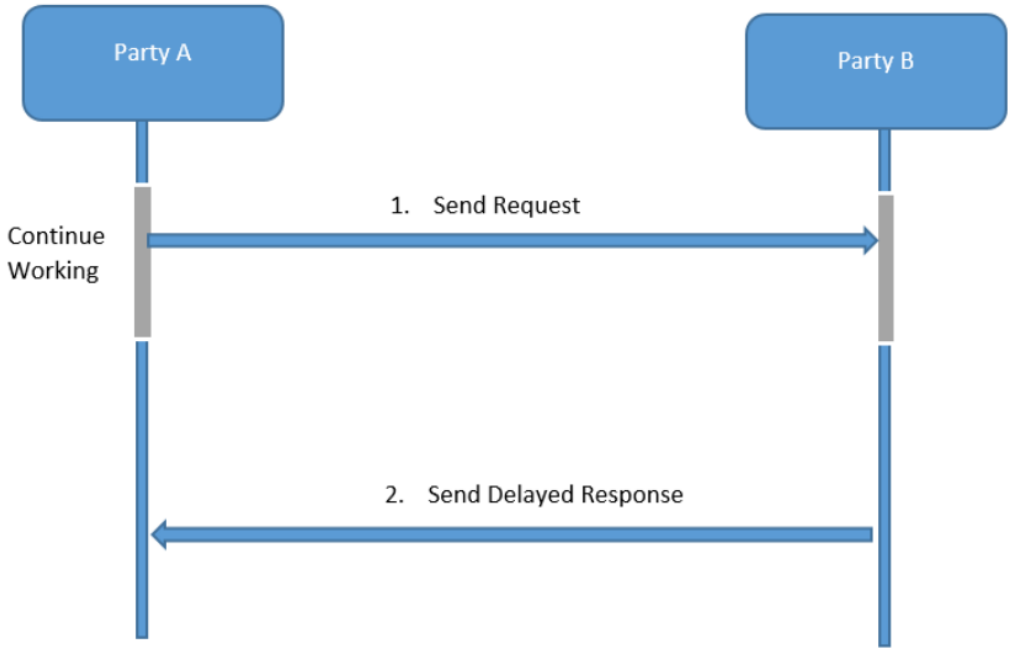


Figure 2.7: Asynchronous message between party A and party B [5]

in this kind of communication since the messages themselves must be compatible across implementations.

Synchronous message: According to the Figure 2.8, when customers are waiting for a service to respond. The HTTP protocol and a stateless client-server architecture are used in REST (Representational State Transfer). It is necessary to utilize this protocol because in a distributed environment, every function is represented by a resource that may be used to carry out activities

Service Discovery

The list of operating service instances in a microservices application evolves dynamically. Service discovery acts as a guide for microservices by maintaining a list of services on which nodes are situated. As a result, in order for a client to issue a service request, the client must utilize a service discovery method. The service registry is a critical component of service discovery.

Management

Due to the complexity of microservices architecture, it may be challenging to maintain load balancing appliances current with feature flags. As a result, a real-time

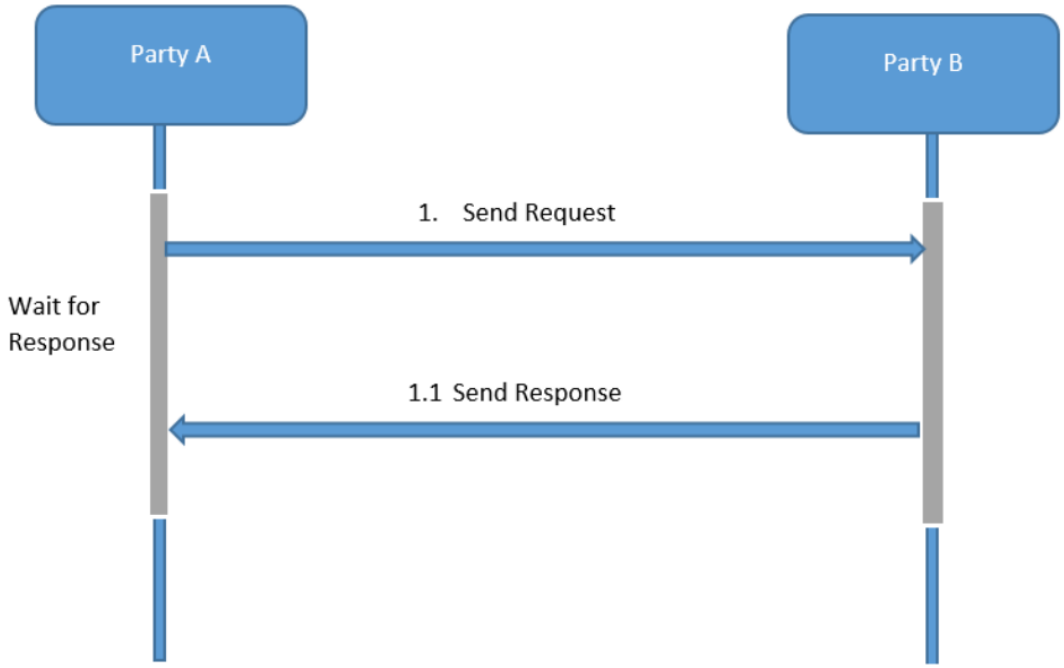


Figure 2.8: Synchronous message between party A and party B [5]

service configuration is provided via a management capability, which is accessible to operations and business users.

Static content

Static material is deployed to a cloud-based storage service once microservices have communicated between themselves and may be sent directly to clients through CDNs (Content Delivery Network).

Database

In order to store their data and execute their business functions, microservices have their own private database. Microservices' databases can only be changed by using the service API. The services provided by microservices are carried forward to any remote service which supports inter-process communication for different technology stacks.

2.1.6 Real-world Case Study of Microservice Architecture

To deal with complicated system management issues, today's world's top companies have moved from conventional architecture to one built on microservices. Microser-

vice architecture is becoming more popular and reinforced for its benefits, as world-leading companies like Netflix, Amazon, Coca-Cola, Spotify, and Uber have made a move from monolith to microservices, paving the way for other companies to follow in their footsteps [41].

Case study of Amazon Microservices

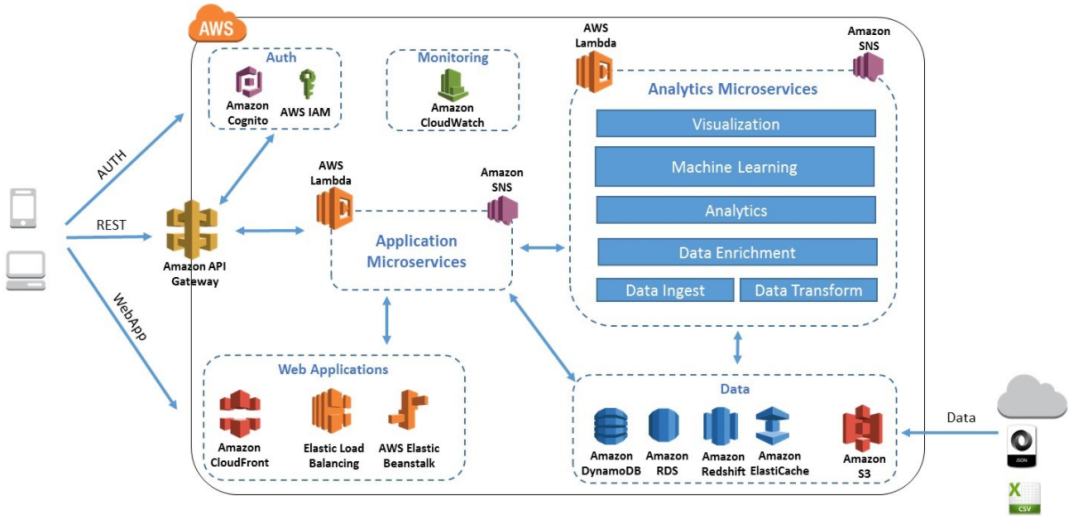


Figure 2.9: Amazon microservices case study [6]

One of the most significant examples of this is the adoption of microservices by Amazon as in Figure 2.9. Amazon is one of the earliest examples of how microservices played an essential part in changing the company. Before Amazon, microservices were not used. Due to the monolithic nature of the old design, all of its functions and components are tightly coupled with one another. Consequently, as the Amazon business became more prominent and included hundreds of engineers, Amazon was no longer able to deliver updates quickly. This claim was supported by Brigham [42], who noted a vital fact at Amazon's re:Invent 2015 conference about what led Amazon to its microservices architecture development. He stated that working with a monolithic application isolates team members and distances the group from its end objective. Therefore, it is necessary for the company to switch from monolithic to microservices. The process began with Amazon using the Elastic Container Service, and Docker, Bridestory's Chief Technology Officer Doni Hanafi, worked with his team to divide the application into more minor services [43]. Currently, the operations of the software development life cycle

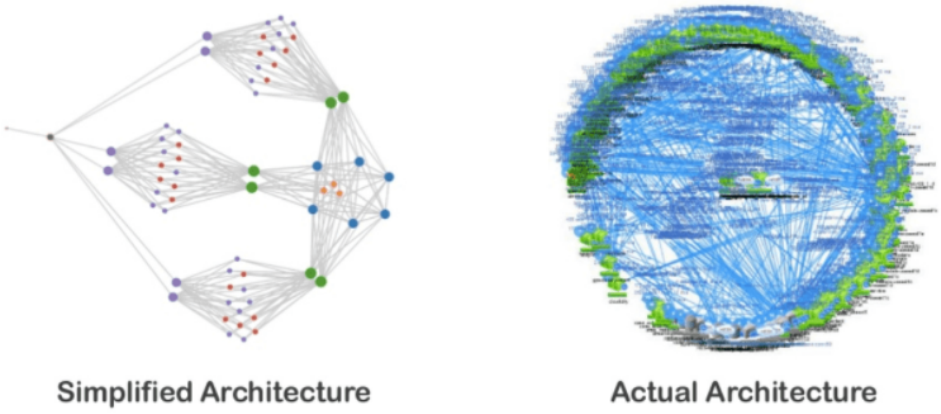


Figure 2.10: Netflix microservices comparison to the simple architect case study [7]

(SDLC) have changed significantly due to the new microservices architecture. According to Hanafi, in the past, deploying a significant feature took the Amazon developer team around three weeks. Now it just takes us a few hours, with the failure rate being less than one percent.

Case Study of Netflix Microservices

Furthermore, to illustrate how microservices may be used to deal with scale problems, consider Netflix as one of the most obvious case study examples as in the Figure 2.10 [44]. Towards the end of 2008, a three-day outage was caused by increased demand on Netflix's infrastructure. Thus, Netflix refactored its monolithic software for these purposes. According to the case study [44], to begin migrating from monolithic to microservices. Netflix started by denormalizing their data model using NoSQL databases and went from a single monolithic app to several microservices. Then Netflix's developer team moved all applications that didn't directly serve customers, including video encoding and other back-end functions. This was followed by a division of customer-facing elements such as account creation, movie choice, device choice, and configuration. Netflix divided its monolith into microservices over two years, and in 2011, Netflix announced the completion of its reorganization and the use of microservice architecture across the company. Remarkably, nowadays, Netflix's application now utilizes over 500 microservices and API Gateways to handle over 2 billion API edge queries per day.

Figure 2.11: Monolithic Architecture of Uber [8]

Case Study of Uber Microservices

In the case study of Uber, the application and microservices architecture are notably intriguing as well [45]. When Uber initially arrived on the market, they only intended to serve one city with their service. With time and increased revenue, the monolithic system started to create problems with scalability and integration. Uber subsequently decided to shift their whole global IT infrastructure over to microservices as a consequence of this occurrence.

According to the Figure 2.11, a single framework was utilized to integrate all of Uber's trip management and driver administration services, including passenger invoicing, notification features, and payments. Thus, the whole system is based on a MySQL database for data storage, and three different adapters with APIs are utilized to do invoicing, payment, and email/message sending.

According to Figure 2.12, Uber's new design includes an API Gateway and separate services, each with a specific function. These services may be created and developed on their own. Now, these modules each have their own unique set of capabilities and

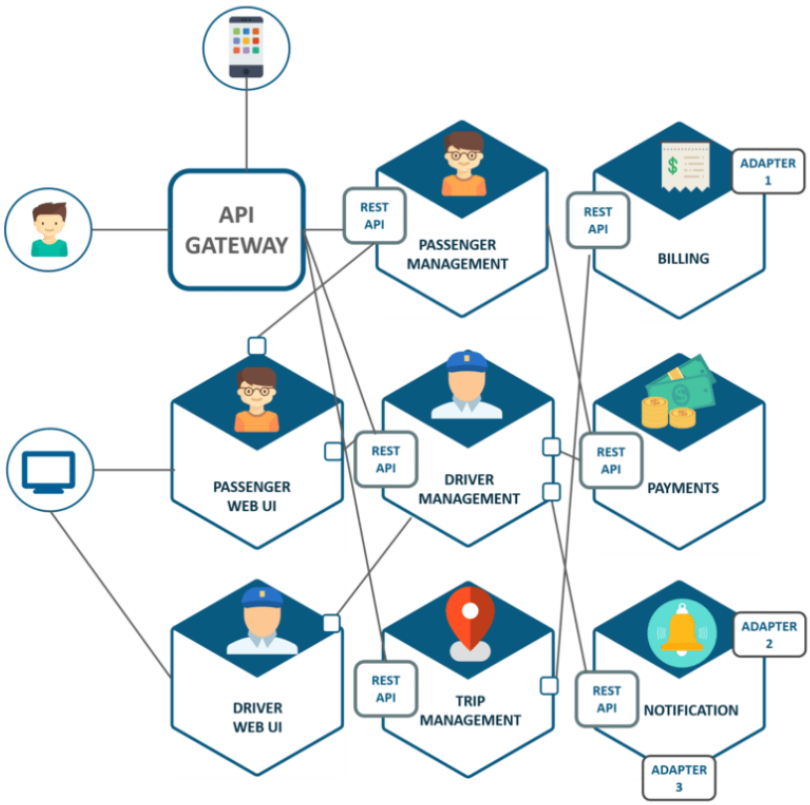


Figure 2.12: Microservices Architecture of Uber [8]

may be used independently. Since each feature could now be scaled separately, the interdependency between them was also removed. Furthermore, the new design of Uber allows the re-implementation of API connection, known as the API Gateway, which links all the cars and passengers is also made possible by microservices.

Therefore, when comparing two architectural software approaches, several studies, for instance [46], [47], and [36], have been carried out to investigate the performance of monolithic software architecture. It has now been suggested that use microservices. Businesses can manage big codebase applications in a more practical manner, with small teams making incremental changes to their own codebase and deploying them to other locations [46], in accordance with the statement from Francesco and his colleagues in the research paper about architecting microservices, which highlights the importance of microservices. Francesco [36] demonstrates that microservices concepts are applicable to a wide variety of businesses. Moreover, the recent article by Md Kamaruzzaman [47], lead software architect, about the overview of microservices also analyzed that, al-

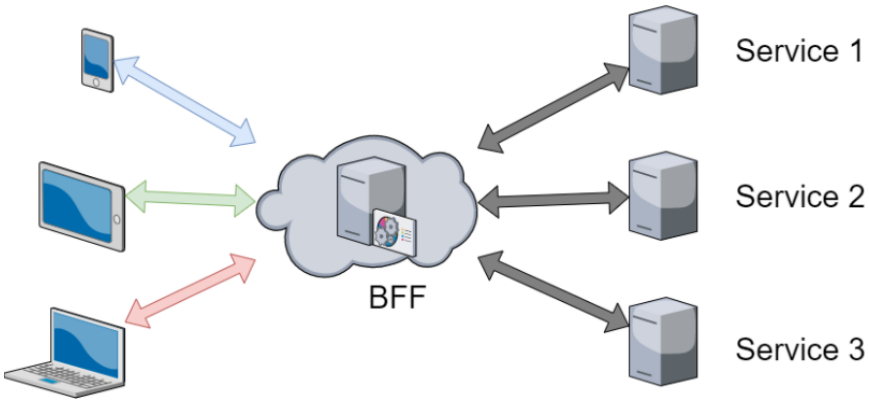


Figure 2.13: Backend for frontend design pattern [9]

though the implementation of microservices may be complex, the concept itself remains the same. Nevertheless, sustainable software development should make the advantages of microservices architecture worthwhile for long-term benefits.

In conclusion, based on the literature review, microservices architectures would see upward trends and widespread adoption in the industrial sector in the not-too-distant future. Moreover, with thorough research through data gathering and literature review, our team has found the advantages of microservices software architecture implementation.

2.1.7 BFF (Backend For Frontend)

A microservices design enables teams to rapidly iterate and create scalable technologies. Figure 2.13 show the Backend for Frontend (BFF) architecture is a microservices-based design. One of the most important parts of this design pattern is an application that links the application's frontend and backend [48]. Using the BFF pattern, data is fetched more efficiently between clients (browsers, apps, and other Internet-connected devices) and services hosted on the servers.

Currently, a tendency toward web-delivered interfaces, rather than thick-client applications, has emerged with the rise of Software as Service Solutions (SASS) due to the popularity of the web and its subsequent success [17]. However, now a challenge and problem have emerged. Since most of the functionality to expose was on the server-side, it must be available through a desktop web UI and one or more mobile UIs. Therefore, most systems, which were built with desktop-web UI in mind, often had issues adapting

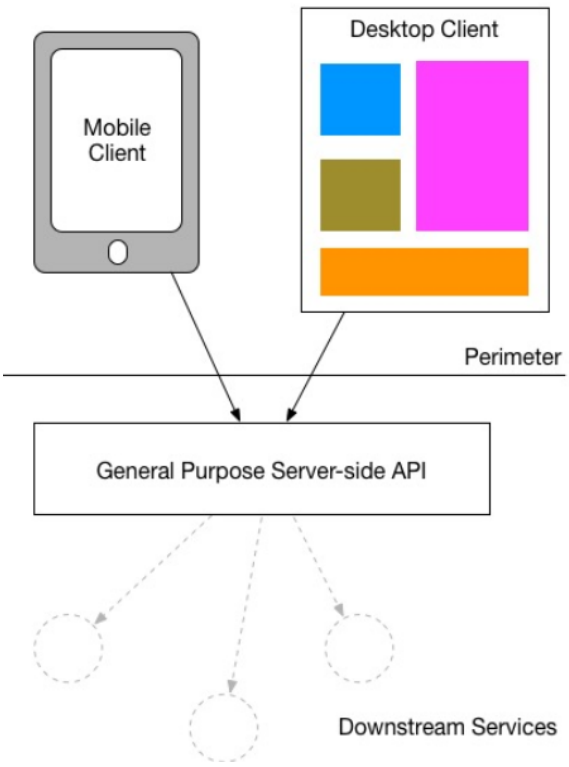


Figure 2.14: A general purpose API backend [10]

to new kinds of user interfaces due to the fact that desktop-web UI and the supported services were already tightly coupled. One of the most prevalent issues is that different teams work on their various frontend application interfaces; all it takes is a single bottleneck on the backend to derail development if it cannot meet the application's demand. And, as every developer is well aware, this complexity, this potential for mistakes and misalignment, will always result in a degraded user experience.

Normally, a traditional way to solve this problem is to use "The general-purpose API backend." Starting by implementing the general-purpose API backend, a single server-side API is often provided, with more functionality added over time to enable new mobile interaction patterns [10].

According to the Figure 2.14, there is a need for a system interface design for the mobile client and desktop client. Since every program needs a fundamental building block, the general-purpose API backend provides such blocks because the general-purpose API backend offers pre-built functionality so that developers can concentrate on creating high-level, high-value business functions instead of implementing the same

basic features over and over again for each new application.

However, there is a problem with the traditional approach of a general-purpose API backend. In an ideal way, if the various user interfaces can implement the same or very similar kinds of calls, then this kind of general-purpose API can be successful. Despite that, in the real-world scenario, there are several significant differences between using the internet on the phone and using it on a computer. For instance, the capabilities of a mobile device vary significantly from those of a desktop computer. On the mobile application, the amount of data can only be displayed in a smaller amount. Since mobile screens are typically smaller in size compared to desktop screens, Thus, mobile devices will want to make different calls, make fewer calls, and show additional (and likely less) data than their desktop equivalents. This requires us to extend the functionality of the API backend to serve the mobile interfaces, which is where the conventional approach of a general-purpose API backend falls short.

To address these problems, a SoundCloud engineer named Phil Calçado developed the Backends for Frontends (BFF) architectural pattern in 2015 [49] as in Figure 2.15. At the time, SoundCloud consisted of a single API that supported both official applications and third-party integrations (the Public API). As a result, a transition from a monolithic design to a microservices architecture was required [50]. Additionally, the development of a monolithic private/internal API and the proliferation of additional microservices has provided new possibilities for the construction of new frontend APIs on top of the monolith. Consequently, authentication, rate restriction, header sanitization, and cache management were all handled by BFF services at SoundCloud. BFFs take all of the external traffic that comes into the data centers, and the BFFs can process hundreds of millions of requests each hour.

Because the BFF is tightly coupled to a specific user experience, it is typically maintained by the same team as the user interface, which simplifies the process of defining and adapting the API as the UI is requires, while also simplifying the process of aligning both the client and server components' releases.

According to the 2.16, in order to take advantage of BFFs implementation and tackle the issues, this can be done by providing each user interface with a separate backend server. Now, with the implementation of BFF, the developer can focus on improving

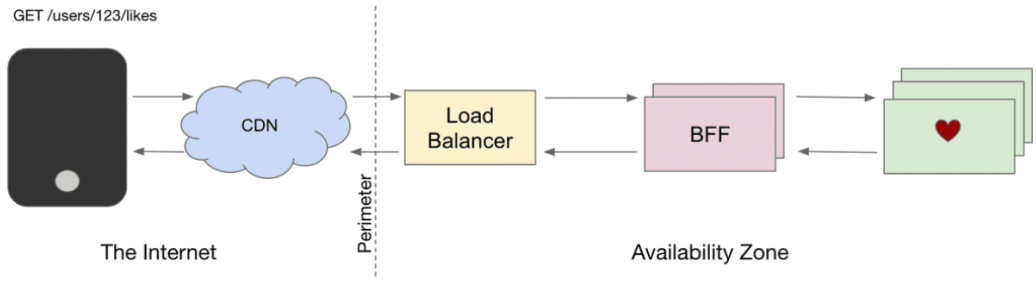


Figure 2.15: BFFs at SoundCloud in 2021 [11]

the frontend experience without impacting other frontends by fine-tuning the behavior and performance of every backend.

Additionally, since each backend is devoted to a particular interface, optimization for that interface is feasible. As a result, it will be more compact, simpler, and likely faster than a generic backend that tries to satisfy all interface requirements. For BFF designing patterns, it is suitable to use this design pattern when the backend needs to be improved to meet the needs of specific client interfaces. Also, BFF design patterns should be used when it is preferable to utilize a separate programming language on the backend of a different user interface or when backend services, whether shared or general-purpose, must be kept current with a considerable amount of development effort [51].

There are many advantages of utilizing BFF design patterns. One of the most notable advantages of the BFF design pattern in this instance is its autonomy. The BFF architecture reduces the time required to make modifications and enhancements to back-end systems by using specialized teams. For example, one group is often responsible for the front end. At the same time, another is responsible for the back end, and even when microservices are introduced, the bigger work is broken down into smaller jobs. However, if a business sector reaches a specific size or complexity, it may face challenges that need the cooperation of several teams. As a result, groups responsible for developing user interfaces sometimes struggle to communicate with an API managed by a different team. This is where the BFF can help; although the BFF must continue to share with other downstream services, this may be accomplished without interfering with UI development. BFF enables parallel development of the BFF's API and the front end, allowing for fast iteration. As a result, developers may optimize the APIs for the

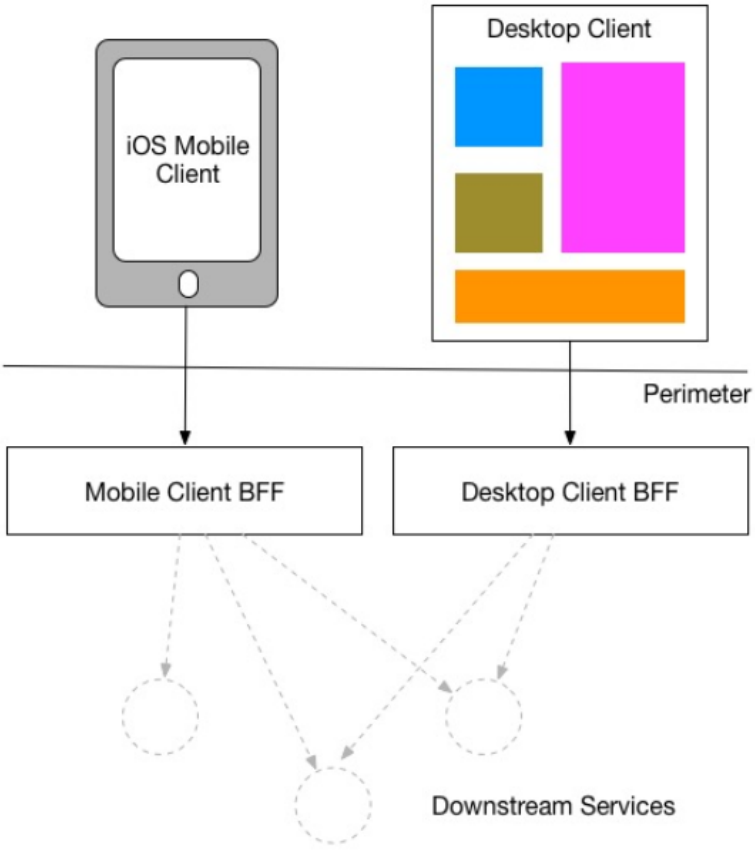


Figure 2.16: BFF per user interface [10]

convenience of each client type without the need for synchronization points or difficulty.

Apart from those mentioned above, BFF enhances the high pace of development. Rapid development is facilitated by great autonomy and minimal risk. Another significant benefit of BFF designing patterns is that BFF controls who may use or modify the API calls. Hence, it makes BFF able to solve the issue of maintaining previous versions of the API on hand for a limited fraction of your external partners who can't make changes using a general-purpose API backend. Also, BFFs also have the advantage of being resilient. Some failed deployments may bring down a real BFF in an availability zone, but this will not affect or bring down the entire platform.

Finally, Backend For Frontend is a design pattern built with the user and their experience in mind, not only the developers. In order to meet, engage, and service consumers, it is the solution to the ever-growing adoption of apps that provide consistency

while fulfilling their varied and changing requirements. As a result, it separates the backend from the frontend. This is a huge step forward for the microservices software designing pattern of the future.

2.1.8 API Security

OWASP Top 10 API Security

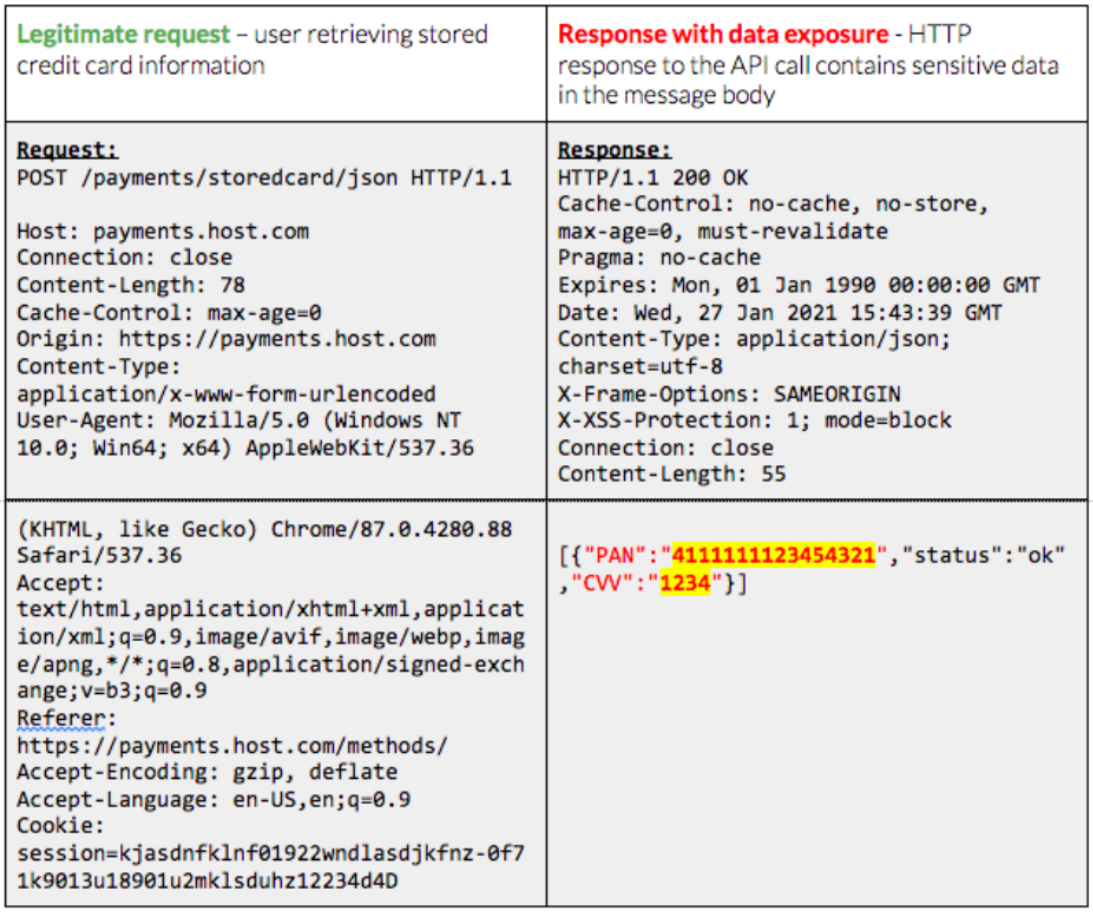
The OWASP Top 10 is a security awareness guide for web developers. The OWASP Top 10 was developed by the Open Web Application Security Project (OWASP® Foundation), a non-profit online community. The OWASP Foundation publishes a variety of material, including research papers, methodologies, tools, and technologies. To improve application security and to offer developers' tools and resources [52].

As of 2003, the OWASP Top 10 project has successfully served as the definitive resource for learning about common online application security vulnerabilities, as well as mitigation strategies. However, now there is a greater threat, which is the API security problems. APIs are critical in today's app-centric environment for driving innovation. Customers, partners and internal applications all use APIs. APIs may be found in a wide range of mobile, SaaS and online apps from banking to retail to transportation, to the Internet of Things (IoT), autonomous vehicles (AVs), and smart cities (Sc) [53]. As a consequence, OWASP began work in 2019 on an API security-specific version of their Top 10. As of December 31, 2019, the OWASP API Security Top 10 list has been published.

Our project will emphasize two aspects in this project's investigation including API 3:2019 — Excessive data exposure and API 8:2019 — Injection

API3:2019 — Excessive data exposure

Generally, when it comes to APIs, it's possible that they disclose a lot more data than the client really needs. As a consequence, in order to take advantage of Excessive Data Exposure, a hacker simply sniffs the traffic and analyzes the API replies, searching for sensitive data that should not be returned to the user [12] . Account numbers, email addresses, phone numbers, and access tokens are all examples of information that may be included.



Legitimate request – user retrieving stored credit card information	**Response with data exposure** - HTTP response to the API call contains sensitive data in the message body
Request: POST /payments/storedcard/json HTTP/1.1 Host: payments.host.com Connection: close Content-Length: 78 Cache-Control: max-age=0 Origin: https://payments.host.com Content-Type: application/x-www-form-urlencoded User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36	**Response:** HTTP/1.1 200 OK Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: Mon, 01 Jan 1990 00:00:00 GMT Date: Wed, 27 Jan 2021 15:43:39 GMT Content-Type: application/json; charset=utf-8 X-Frame-Options: SAMEORIGIN X-XSS-Protection: 1; mode=block Connection: close Content-Length: 55
(KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 Referer: https://payments.host.com/methods/ Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.9 Cookie: session=kjasdnfklnf01922wndlasdjkfnz-0f71k9013u18901u2mklsduhz12234d4D	[{"PAN":"4111111123454321","status":"ok","CVV":"1234"}]

Figure 2.17: API 3:2019 Excessive Data Exposure case study [12]

Developers often attempt to build APIs in a general manner without considering how sensitive the data they expose could be. As a result, this kind of vulnerability may be detected by traditional security scanning and runtime detection technologies from time to time. Furthermore, with a lack of API security implementation, sometimes the developer cannot tell the difference between legal API data and sensitive data that should not be provided. Undoubtedly, in-depth knowledge of the application's design and API context is needed for this.

According to the Figure 2.17, API 3:2019 Excessive Data Exposure case study, a POST request is sent from the user's web browser to a backend API in order to retrieve previously saved payment information. Specifically, the API is obtaining the primary account numbers (PAN) and CVV code from a previously saved credit card transaction. In the example above, x-frame-options are used to defend against cross-frame scripting attacks, while the x-xss-protection header is used to prevent cross-site scripting assaults.

However, relying only on client-side programming to filter or conceal such sensitive data isn't enough. Nowadays, attackers frequently circumvent client-side web application and mobile application code, making API calls directly and exposing sensitive data.

This failure is likely to happen because the Web application firewall (WAF) and API gateways are examples of traditional security measures that do not comprehend the exposure risk of the information they are sending. While these kinds of filters may capture well-defined sensitive data types such as PANs or social security numbers (SSNs), they do not comprehend the API context and business logic flow.

Case Study on Excessive Data Exposure

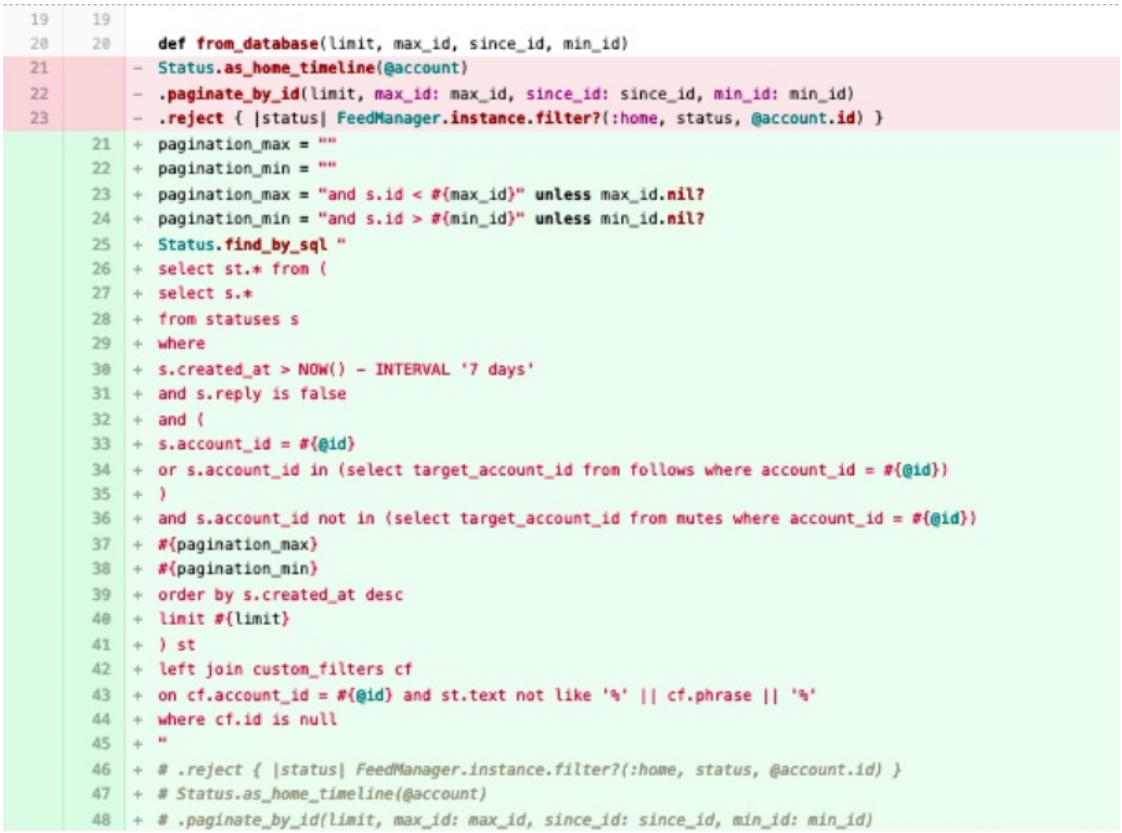
According to a real-world case study of the Flaw business [54], which exposed the personal data of 2 million Bounceshare users to hackers. Consequently caused by excessive data exposure. In 2019, the Bounce Share app posed a security concern to users. However, when a phone number is provided in an API request, a security researcher discovered that the Bounceshare app returns an access token and RiderId linked with the account for that particular phone number.

As a result, an attacker might combine a phone number dump and a script to get unauthorized access to many user accounts. A hacker who has obtained access to a target user's Bounceshare account will have complete access to the victim's personal information, such as their driver's license, email address, and even photos. Also, attackers can even access user payment account balance and achieve the capability to book a ride by using this excessive data leakage.

In conclusion, the API security solution should be able to detect and report on the wide range of sensitive data types that may be provided in API requests and replies to be effective. Additionally, these systems must monitor API access by endpoint and individual users to detect the misuse of personal data. To avoid false positives or denied requests, these solutions must also offer API context and a wide variety of response behaviors.

SQL Injection

SQL Injection is a type of SQL attack that tries to find, insert, change or delete unauthorized data by utilizing special characters in SQL syntax to cancel the original



```
19 19
20 20   def from_database(limit, max_id, since_id, min_id)
21      - Status.as_home_timeline(@account)
22      - .paginate_by_id(limit, max_id: max_id, since_id: since_id, min_id: min_id)
23      - .reject { |status| FeedManager.instance.filter?(:home, status, @account.id) }
   21   + pagination_max = ""
   22   + pagination_min = ""
   23   + pagination_max = "and s.id < #{max_id}" unless max_id.nil?
   24   + pagination_min = "and s.id > #{min_id}" unless min_id.nil?
   25   + Status.find_by_sql "
   26   + select st.* from (
   27   + select s.*
   28   + from statuses s
   29   + where
   30   + s.created_at > NOW() - INTERVAL '7 days'
   31   + and s.reply is false
   32   + and (
   33   + s.account_id = #{@id}
   34   + or s.account_id in (select target_account_id from follows where account_id = #{@id})
   35   + )
   36   + and s.account_id not in (select target_account_id from mutes where account_id = #{@id})
   37   + #{pagination_max}
   38   + #{pagination_min}
   39   + order by s.created_at desc
   40   + limit #{limit}
   41   + ) st
   42   + left join custom_filters cf
   43   + on cf.account_id = #{@id} and st.text not like '%' || cf.phrase || '%'
   44   + where cf.id is null
   45   + "
   46   + # .reject { |status| FeedManager.instance.filter?(:home, status, @account.id) }
   47   + # Status.as_home_timeline(@account)
   48   + # .paginate_by_id(limit, max_id: max_id, since_id: since_id, min_id: min_id)
```

Figure 2.18: SQL Injection example case [13]

query and replace it with the attackers. The SQL injection example is shown in Figure 2.18

The notable exploitation that used SQL injection occurred in March 2021, where Gab was breached by SQL injection, and 70 gigabytes of data were downloaded. This fault was introduced by the CTO of Gab, Fosco Marotto, who made a git commit that replaced parameterized query with raw SQL statement shown in Figure 2.19.

To explain, The school may have this statement to insert children's names into their database. `INSERT INTO Students (firstname) VALUES ('Pumipat');` But as the mom named her son `Robert'); DROP TABLE Students;--` the statement has become `INSERT INTO Students (firstname) VALUES ('Robert'); DROP TABLE Students;--');` the statement is now turn into three statements as follows;

```
INSERT INTO Students (firstname) VALUES ('Robert');
DROP TABLE Students;
-- ');
```

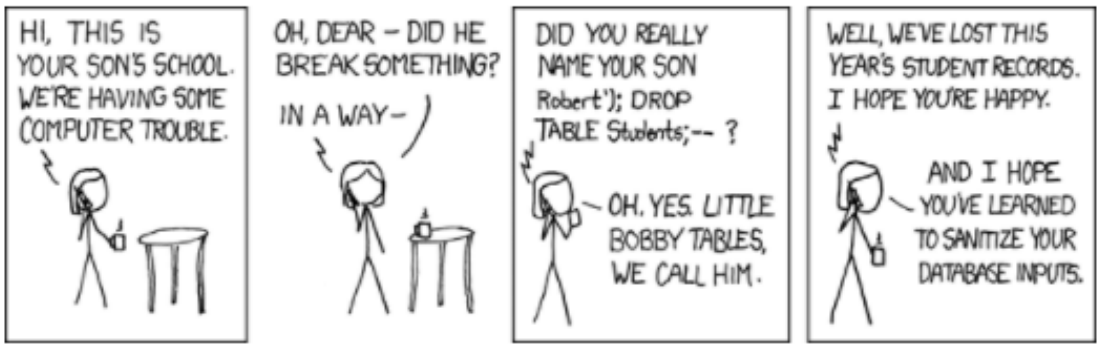


Figure 2.19: SQL joke [13]

When the first one inserts the student's name as usual, but the second statement removes the student's table from the system, and the original closing quotation mark is now negated by being marked as a comment. After the statements were executed, the school's database lost the student record.

2.1.9 API Testing

API testing is a technique for validating the API's operation. The purpose of testing is to establish whether the functionality, security, reliability, and performance of an API adhere to given requirements [55]. The most often used way of testing is to request the API and verify that it works appropriately according to the logic. This may be accomplished manually or automatically. However, manual testing is not advised since it requires the tester to enter each API and record the result manually. If the inputs are laborious and the APIs vary, the tester's job becomes a burden. Despite that, the methods of automated testing, on the other hand, are increasingly suggested. Now more than ever, API automation testing needs software capable of making and receiving requests, which the tester may either use a testing tool or develop their automation code to their software.

There are many advantages to API testing. Firstly, after the logic has been developed, API testing is used to verify the accuracy of replies and data. Thus, developers do not need to wait for different teams to complete their work or for whole apps to be developed anymore since test cases are isolated and available to be built immediately. Secondly, the API enables more straightforward test maintenance. For instance, user interfaces are continuously evolving and shifting in response to how users view items. Therefore, this results in a nightmarish situation in which tests are constantly rewritten

to keep up with the production code. Now, API testing may assist in automating the process of refactoring tests within seconds. Thirdly, APIs provide a more rapid time to resolution. Because when API tests fail, developers will be able to know precisely where the system went wrong and where to look for the problem. This helps to minimize time spent triaging issues across many builds, integrations, and even team members. In addition to this, the tiny, isolated footprint of an API test is ideal for calculating quicker mean time to repair (MTTR), a critical KPI for DevOps teams. Fourthly, API testing is very efficient in terms of speed and coverage. Three hundred user interface tests may take up to 30 hours to perform. Three minutes may be used to perform 300 API tests. This means more problems are discovered in less time.

2.1.10 API Fuzzing

Nowadays, Some of the world's largest and most reputable companies, such as Google, Microsoft, and the US Department of Defense (DoD), are using fuzzing in their quality control and cybersecurity operations [56]. OWASP describes 'Fuzzing' or 'Fuzz testing' as a method for doing black box software testing [57]. Fuzzer will inject random input into the target program automatically and will identify the problem. The name 'The Fuzzer' has come to refer to a fuzzing testing instrument equipped with a generator capable of generating combinations of input types such as integer, string, char, and so on.

Using fuzzing to discover security flaws and vulnerabilities in software is an effective and common practice. A target program is fed with erroneous test data to see whether there's any vulnerability in the execution. The original random fuzzing system has been enhanced by combining several useful techniques, including dynamic symbolic execution, coverage advice, grammar representation, scheduling algorithms, dynamic taint analysis, and machine learning [58].

To begin, the Fuzzing methodology needed a predetermined list of inputs that might result in invulnerability, as is customary for this method. Following the method definition, the Fuzzer will create the request using the specified input and submit it to the APIs. The Fuzzer will then wait for a response to each request and verify the answer against a supported rule to determine whether the request and response violated any rules.

After the test is complete, the tester will review the results and forward the problem for further investigation.

There are many types of Fuzzer, such as Mutation-Based Fuzzers, Generation-Based Fuzzers, and Protocol-Based Fuzzer. One of the most well-known Fuzzers that has a comprehensive knowledge of the protocol format being tested is known as Model-based test development, which involves loading a specification array into the tool and then parsing the specification for abnormalities in the data contents sequence, and so on. Structural testing is also known as language testing, robustness testing, and similar names. Fuzzer may use valid or invalid inputs, or they may construct test cases entirely from scratch.

Fuzzing techniques can be used to detect three types of bugs[59]. Firstly, a fuzzer can detect Assertion failures and memory leaks, which happens when a computer application fails to release unused memory because the allocations are handled improperly. This technique of detecting this type of bug is frequently employed in large systems where memory safety issues exist, making them a serious vulnerability. Secondly, it can detect an invalid input, which is done by using the technique to generate invalid input to test error-handling methods. Thus, negative testing may be done automatically using a fuzzing approach. Thirdly, fuzzing techniques can be used to detect correctness bugs. In the fuzzing technique, it's possible that fuzzing may uncover "correctness" issues in certain situations. An example of this would be a corrupt database or poor search results.

There are many advantages and disadvantages of Fuzzing techniques. On the one hand, the fuzzing techniques can provide some benefits, such as it can enhance software security testing. Also, if the testers overlook any issues because of a lack of time or resources, such problems will be discovered during fuzz testing.

On the other hand, fuzz testing on its own cannot provide an incomplete view of an entire security threat or issue. Additionally, only minor flaws or risks may be detected with fuzz testing. When cybercriminals want to do the greatest harm in the quickest time possible, often use the strategies like Buffer overflow, DOS, cross-site scripting, and SQL injection, vulnerabilities may all be found with fuzzers. However, there is a limitation with fuzzing tools since they cannot detect the vulnerabilities that caused a program crash. For instance, when spyware, viruses, worms, Trojan horses,

and keyloggers are tested using fuzzing, the fuzzer cannot detect them [60]. Apart from that, fuzzer requires the random input to make a boundary value condition, which this process could be complicated.

In conclusion, fuzz testing is used in software engineering to determine the existence of flaws in an application. Fuzzing cannot ensure that all deficiencies in a program are detected fully. However, by using the fuzzing method, the application is made more resilient and secure since this technique exposes the majority of frequent flaws.

2.1.11 The Significance of HTTP Error Code 500

Numerous 500 Internal Server Errors are just the result of an implementation's lack of error handling. Some resource requests may be lacking validation. Certain types of input may fail to deserialize appropriately, for example, when the path or query parameter does not match the intended variable type, uncontrollable by the developer factors like failed infrastructure.

There are many ways for sensitive data to be leaked. These include both unlawful memory accesses and legitimate memory that includes information that should not be accessible to the public. In this example. To begin, a small amount of program memory that contains both real data and random data was being implemented.

```
secrets = ("<space for reply>" + fuzzer(100)
     + "<secret-certificate>" + fuzzer(100)
     + "<secret-key>" + fuzzer(100) + "<other-secrets>")
```

Figure 2.20: Information leak fuzzer case : uninitialized memory [14]

In this case Figure 2.21, "deadbeef" is an identifier for uninitialized memory by adding extra "memory" characters to secrets.

```
uninitialized_memory_marker = "deadbeef"
while len(secrets) < 2048:
    secrets += uninitialized_memory_marker
```

Figure 2.21: Information leak fuzzer case : uninitialized memory [14]

Then, it is necessary to establish a service that requires a response to be given back, as well as a time limit. When the reply is to be sent, it would keep it in memory and then send it back with the length that was specified as Figure 2.22.

```python
def heartbeat(reply, length, memory):
    # Store reply in memory
    memory = reply + memory[len(reply):]

    # Send back heartbeat
    s = ""
    for i in range(length):
        s += memory[i]
    return s
```

Figure 2.22: Information leak fuzzer case : length specified [14]

With the regular strings, this application will run flawlessly as in Figure 2.23.

```
heartbeat("potato", 6, memory=secrets)
'potato'
heartbeat("bird", 4, memory=secrets)
'bird'
```

Figure 2.23: Information leak fuzzer case : normal case [14]

There will be an extra memory leak if the request is longer than the reply string can handle. It's important to note that all of this takes place inside a standard array and hence does not result in an address cleaner being called as in Figure 2.24.

```
heartbeat("hat", 500, memory=secrets)
'hatace for reply>#,,!3?30>#61)$4--8=<7)4 )03/%,5+! "4)0?.9+?3();<42?=?0<secret-certificate>7
(+/+((1)#/0\'4!>/<#=78%6$!!$<-"3"\'-?1?85!05629%/); *)1\'/=9%<secret-key>.(#.4%<other-secrets>
deadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbe
efdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdead
beefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefde
adbeefdeadbeefdeadbeefdeadbeefdeadb'
```

Figure 2.24: Information leak fuzzer case : invalid case [14]

Because of this, it is feasible to discover information leakage issues. Because identifying information that should not be released, such as the given secrets and uninitialized memory, is essential. By using a simple example, it is shown that the developer is capable of doing a check of this kind. Fuzzing, as previously mentioned, may be used to stop API data from leaking. Because it might alert the developer to data leaking inside the system, status error code 500 is critical as in Figure 2.25.

```
from ExpectError import ExpectError

with ExpectError():
    for i in range(10):
        s = heartbeat(fuzzer(), random.randint(1, 500), memory=secrets)
        assert not s.find(uninitialized_memory_marker)
        assert not s.find("secret")
```

```
Traceback (most recent call last):
  File "/var/folders/n2/xd9445p97rb3xh7m1dfx8_4h0006ts/T/ipykernel_13728/4040656327.py", line
4, in <module>
    assert not s.find(uninitialized_memory_marker)
AssertionError (expected)
```

Figure 2.25: Information leak fuzzer case : status error code 500 case [14]

In conclusion, error code 500 is used in the negative testing, which is often overlooked by the developer and quality assurance tester. Since the majority of testing attention is concentrated on positive testing, and only ensuring an API behaves correctly on the positive route. This is often where difficulties develop, since there are no procedures in place to deal with unfavorable events. Fuzzing checks use a variety of ways for determining whether a defect vulnerability has been found. Additionally, assertions may be disabled and customized. For instance, the API fuzzer by default makes use of HTTP status codes to determine whether there is a genuine problem. A fault is created when an API returns a 500 error during testing. This is not always desirable, since some frameworks often return 500 errors. As a result, it is critical to understand the error 500 status code. API Fuzzing comes to the rescue in this situation. Through negative testing, API developers may attempt to identify these vulnerabilities early.

2.2 Tools and Techniques

This section explains several tools and techniques which are applied and used in this project.

2.2.1 Node.js

Node.js is a platform built on top of Chrome's JavaScript engine that enables the rapid development of high-performance, scalable network applications. Node is the JavaScript language's server environment. Moreover, Node runs JavaScript on Google's V8 engine and makes system calls using the self-developed libuv library. Additionally, Node. Js is lightweight and efficient because of its event-driven, non-blocking I/

O style, ideal for data-intensive real-time applications that operate across dispersed devices. Time was a significant factor in Node.js's progress. Only a few years before, JavaScript gained recognition as a more severe language as a result of "Online 2.0" applications such as Flickr, Gmail, etc. Previously, Netscape created JavaScript as a programming language for Netscape Navigator, the company's Web browser. Netscape's business model was based on selling Web servers equipped with a server-side JavaScript environment called Netscape LiveWire. Although Netscape LiveWire had some early success, server-side JavaScript did not gain widespread use until the mid-nineties with the debut of Node.js [61].

Node.js founder Ryan Dahl intended to build real-time websites with push functionality, "inspired by services like Gmail." He provided developers with a tool for working in the non-blocking, event-driven I/O paradigm with Node.js [62]. Based on the article from Procoder [63], frequently, a Node is built as a "microservice" architecture, with each Node offering a self-contained route to execute a particular service. It's a novel way to break down an application into its basic components. However, it is a very successful approach to administering mobile applications, which need the highest levels of speed, accessibility, and accuracy.

There are several benefits of Node.js. Firstly, one of the most important key advantages of Node.js is that it supports asynchronous operations [64]. Node processes JavaScript scripts using the V8 engine, and its most important characteristic is a single thread running, which can only do one operation at a time, which means that tasks aren't instantly completed but are instead added to a task queue and executed when the one before it is complete. As a result of this feature, tasks are often specified as callback functions. As a result, Nodes.js makes this ecosystem valuable for programmers, regardless of whether the developer is creating applications for the iPad, iPhone, or Android. Tasks may execute concurrently and smoothly without slowing down the server or taking up too much capacity since they are divided into distinct, separate node pathways.

Secondly, the capacity to rapidly expand is another advantage of Node.js [63]. he reason for this is because Node.js nodes are built around "events." This implies developers may add new resources to their current programming while also scaling it vertically and horizontally. There are no practical limits to how many components a computer

program may include, and scalability, in any case, allows a program to expand.

Thirdly, high speed and performance are other advantages of Node.js [63]. Because of the non-blocking input-output procedures provided by Node.js, it is one of the fastest alternatives. Runtime performance is improved because code is executed more rapidly, and this is primarily because of the system's partitions. Fourth, efficient caching, Node.js is capable of storing a large quantity of data in a small amount of memory [63]. In-app memory is used to cache requests made to the app. As a consequence, whenever requests are completed and re-executed. Fifth, the ability of Node.js to utilize a single programming language simplifies things. Because JavaScript powers nodes, programmers can easily incorporate them into the rest of the full-stack development process. This allows front-end developers to concentrate on more straightforward back-end coding tasks, and additional server-side languages are unnecessary. While node JS is excellent for most applications, there are a few exceptions, such as those requiring a lot of computational CPU power. However, not every corporate application will benefit from using the flexible runtime environment provided by the environment. The agile, modular architecture, on the other hand, is ideal for startups that need to be flexible throughout the development process. In terms of implementation, Node.js has several benefits for new businesses getting off the ground. Its inherent advantages work nicely with startups' need for flexibility and scalability. Startups will benefit from lower development expenses because of the open-source ecosystem accessible to programmers. However, more than a thousand businesses have placed their faith in Node.js, including some of the most well-known and well-respected names globally, such as Netflix, Paypal, NASA, Uber, and Linked In [65]. In conclusion, while Node.js was fortunate to be constructed in the right location and at the right moment, this is not the sole reason for its current popularity. It presents a slew of novel ideas and methodologies for JavaScript server-side programming that have already proven beneficial to many developers. Since developers may use non-blocking I/O requests and execute on a single thread, which helps the developer to manage tens of thousands of threads within their microservices software in an event loop with Node js.

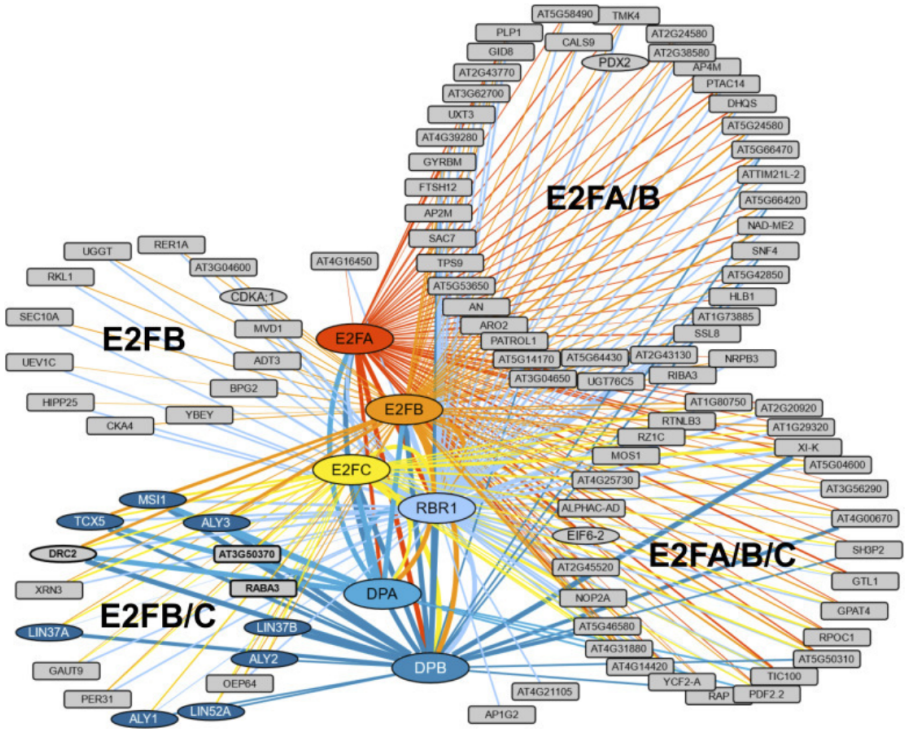


Figure 2.26: The DREAM complex represses growth in response to DNA damage in Arabidopsis, constructed by Cytoscape

2.2.2 Cytoscape Js

Cytoscape is a free and open-source software application. Initially developed for biological research, Cytoscape has evolved into a robust network visualization and analysis tool [66]. Cytoscape visualizes molecular interaction networks and biological pathways via the use of open-source software. As a result, it enables a wide range of applications in molecular and systems biology, genomics, and proteomics. Pathway databases maintained by humans such as WikiPathways[67], Reactome [68], and KEGG [69] may be seen and analyzed in this manner. Cytoscape has also been used successfully in a large number of publications [70]. Lang et al. found that the DREAM complex represses growth in Arabidopsis in response to DNA damage as in Figure 2.26.

However, nowadays, when it comes to visualizing and analyzing substantial social networks of interpersonal interactions, sociologists often utilize Cytoscape [71]. Additionally, several web service APIs and scripting languages collect social media interactions and store them in standard data file formats. Because Cytoscape is compatible with a wide range of file types, because it's domain-independent, Cytoscape is a fantastic

tool for delving deep into complex networks. It's also possible to improve and expand Cytoscape's capabilities. Cytoscape has a robust app development ecosystem, as seen by the more than one hundred third-party apps produced for the platform [72]. According to his introduction to Franz and his team [73], his work is about "Cytoscape.js: a graph theory library for visualization and analysis," highlighted the fact that Cytoscape.js may now be used on both the client and the server since more and more JS code is shared across clients and servers. Owing to Cytoscape.js characteristics, which allow them to be utilized without a graphical user interface when used as a visualization component by employing the HTML5 canvas as its basic implementation. One of the main features of Cytoscape.JS, which makes it become one of the most successful open-source JavaScript-based graph libraries, is due to the simplicity of the library and its functionality. For ease of use, the library adheres to various concepts of the HTML, CSS, JS web paradigm [73]. Cytoscape.js uses stylesheets that resemble CSS and the markup language syntax. Therefore, graph elements can be customized easily using stylesheets and accessible programmatically through the JS core API. The Cytoscape.js architecture consists of two parts: a core and a collection. On the one hand, the core is where a programmer enters a library for the first time. With this object, layouts may be executed, displays can be changed, and many other things can be done. With the help of fundamental functions, you can get at graph components, and these functions return a set of items. On the other hand, a collection is equipped with its API for filtering, navigating, executing actions, and retrieving data. To see a graph, three essential operations must be performed. Consisting of Data is loaded, a style is applied, and the page is rendered using a rendering algorithm [74]. To begin, the developer must populate the system with data. Then, the style dictates the appearance of the graph's components, which is often stored in a JSON file that follows the same principles as the CSS standard. Finally, after the graph has been entirely shown, it may be subjected to various search and analysis tools. For instance, developers may use Cytoscape.JS to produce a graph and then apply Breadth-First Search (BFS) or Depth-First Search (DFS) algorithms to the Cytoscape JS-generated graph. In terms of performance analysis of Cytoscape, Franz [73] mentioned in his study that the visual styles used, the graph size, and the web browser client all affect the efficiency of the rendering process. Allowing developers to display thousands

of graph components on even the simplest processors. However, Franz [73] suggested in his study that utilizing simpler styles, especially for edges, might improve rendering performance. In conclusion, based on the literature review, the Cytoscape.JS library is a powerful tool with numerous possibilities. Creating visualizations, data analysis, and network analysis.

2.2.3 Docker

Cloud-native development and hybrid multi-cloud systems, both of which rely on containers, are gaining popularity among enterprises. According to IBM [18], Docker is a free open source containerization platform that executes code in any environment. Docker programs may be packaged with OS libraries and other required components and delivered as containers. Additionally, Docker is a collection of tools that enables programmers to quickly create, deploy, operate, update, and stop containers using a single API and a few simple instructions. Assisting developers in rapidly deploying production-ready apps by allowing developers to adjust their code for usage in any environment.

Compared to Virtual machines as in Figure 2.27, Docker is a container operating system that provides a standard way to run Docker programming. Therefore, the container acts as a stand-in for the server's operating system. Similar to Virtual Machine Emulation, but more powerful.

One of the most remarkable features of Docker is that it helps make the process of implementing containers become much simpler tasks for developers. There are several advantages to using container technology [18], such as application isolation, cost-effectiveness, and disposability. Apart from this, according to the AWS article [75], another principal advantage of Docker is that Docker containers can create and scale distributed application architectures like microservices. This claim was also supported by Abraar Syed & Karthic Raois [76]. They mentioned a well-known open-source program that constantly improves and has good community support, making it the perfect tool for microservices software developers.

Another advantage is the continuous integration and delivery of Docker. Establishing a consistent environment makes it possible to deploy applications faster since there are no longer any conflicts across language versions or layering systems. Also,

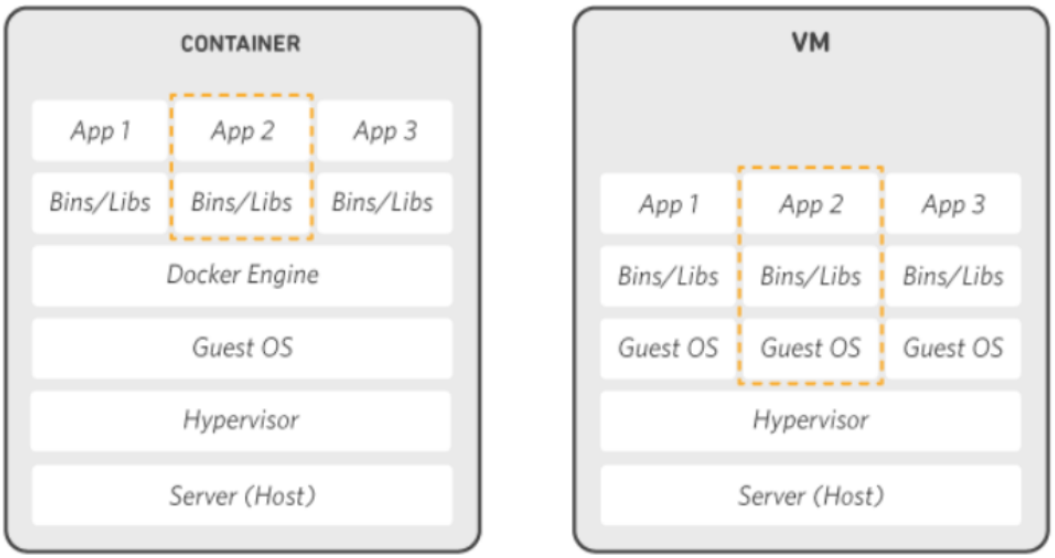


Figure 2.27: Docker and VM comparison [15]

Docker can manage big data processing effectively. With Docker, extensive data processing services and data and analytics are packaged into transportable containers for non-technical individuals to use. Also, Docker has a distinct edge over virtual machines like Hadoop clusters. Since Docker containers are significantly lighter and need less time and effort to set up [77].

From a business perspective, there are several advantages to using Docker. Docker, on average, speeds up software deployment seven times. This means that the product development phase can be significantly reduced, thereby accelerating return on investments and improving long-term profitability as the team can move from project to project more quickly as a result of the increase in productivity. Docker also saves costs in the long run by simplifying the software updates procedures, which also improves profitability.

The previously mentioned benefits are predicted to result in widespread adoption of Docker, which can be seen in the statistical analysis of 415 research firms [77] which anticipates that the container market revenue will expand at a compound annual rate of 35 percents through 2021. Remarkably, Many notable companies are already integrating Docker into their systems, including AWS, Microsoft Azure, Ansible, Kubernetes, and Istio.

In conclusion, containerization of software has never been simpler due to Docker,

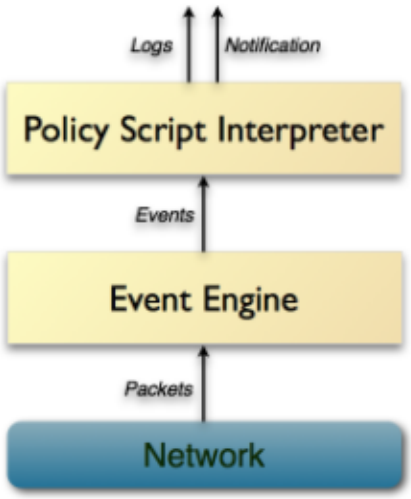


Figure 2.28: Zeek process [16]

the world's leading containerization solution. A Docker container encapsulates complicated microservices, allowing them to be controlled and deployed independently. Because each of these containers will be responsible for a particular aspect of the business's operations, Docker is well suited to the rise of microservices.

2.2.4 Zeek–Network Monitoring Tool

Zeek is the network monitoring tool that can log network activities in a compact and customizable file format. It supports varieties of protocol out-of-the-box and can be added by its programming language. Unlike most network monitoring tools, Zeek uses Turing-complete scripting language targeted for network protocol analysis over signature detection or byte matching. Zeek also runs as free and open-source software which can be an alternative for expensive Intrusion Detection Systems (IDS).

Zeek was initially developed at Lawrence Berkeley National Laboratory by Vern Paxon under the name "Bro" as referencing George Orwell's 1984 novel as the reminder that monitoring can be a privacy violation tool. It was changed in 2019 to Zeek as Bro bears the negative connotation of "bro culture" of the computing world.

Figure 2.28 shows how Zeek works with two components, the Event engine, and Policy Script Interpreter, as the network packet comes in. The analyzer inside the event engine detects the protocol and its activities, and then the policy script decides what to do with each event.

Based on our team's research, our group chose Zeek for its cost and ability to customize and integrate other systems as needed.

2.2.5 RESTler–API Fuzzing tools

RESTler is the REST API Fuzzing tool that was created by Microsoft [13]. RESTler can automatically test APIs for finding bugs such as security and reliability issues. Furthermore, RESTler can automatically generate the test requests from Swagger specification and identify the producer-consumer dependencies, allowing the user to configure these dependencies manually.

Rest APIs are the most popular method of accessing cloud and internet services programmatically nowadays. Web service developers, however, continue to employ static analysis and fuzz testing despite the fact that they are ineffective in protecting the software from an attack. As a consequence, these engineers need automated methods now more than ever to identify problems that may compromise API services, either deliberately by attackers or unintentionally through atypical use patterns. Because of these issues, there have been many recent papers [78],[79], and [80] released by the Microsoft Research team looking at innovative ways to automate the detection of security and reliability issues in cloud/web services using their REST APIs. Subsequently, Microsoft has developed and published new open-source tools to help them regularly test their REST APIs for security and reliability issues.

RESTler is a web-based application that automates the process of testing and detecting security and reliability issues in cloud/web services exposed through their REST APIs. It can operate on 64-bit Windows or Linux computers, and there is also macOS experimental support available. RESTler develops and executes tests that exercise the REST API of a cloud/web service using the OpenAPI/Swagger standard. RESTler uses the API definition to make intelligent inferences about the relationships between request types and then tests for various problems. In addition, developers may incorporate continuous testing into their builds using Microsoft Research's RESTler and a self-hosted REST API fuzzing service. A collection of REST API fuzzing tools can be defined by the developer and hosted.

One of the key features of RESTler is that it is Stateful REST API Fuzzer, which

means that the test case will be executed in the order in which the requests are received. As a consequence of the fuzzing being performed while the request routes are still valid, the service's code coverage is increased, increasing the chance that the unique service code responsible for processing the fuzzed request will be reached. It saves time by avoiding verifying invalid inputs that the service has previously appropriately processed. Another key feature of RESTler is its ability to be extensible, RESTler has numerous built-in security checks that may be activated or disabled during fuzzing, as well as the ability to build customized security checkers and plug them into the fuzzing loop. The last key feature is that RESTler is fully automated, as it generates the fuzzing language from the Swagger/OpenAPI specification. It is pre-configured with enough coverage for basic, well-documented APIs.

RESTler runs in 4 modes, including Compile, Test, Fuzz-learn, and Fuzz [13].

1. **Compile:** Create a RESTler syntax from a Swagger JSON or YAML specification (and optionally samples). As a result of analyzing the Swagger/OpenAPI standard, RESTler generates a fuzzing syntax that includes details about each request's arguments, replies, and interdependencies [81].

2. **Test:** Run a collection of RESTler grammar endpoints and methods fast to see whether they work as expected. This mode is also referred to as a smoke test.

3. **Fuzz-lean:** Rapidly checks potential flaws by running a default set of checkers on every endpoint and method in a built RESTler language.

4. **Fuzz:** Detect vulnerabilities by using the intelligent breadth-first-search mode on a RESTler fuzzing language.

Figure 2.29 shows detail of RESTler process. RESTler begins its process by analyzing the Open API Specification, or Swagger, by creating a pattern for the input as the first step. Then, the second step will pass that input to the API and learn from the responses derived from the previous information. Then Restler will turn it into a tool called "Stateful," as shown in the Figure 2.29.

RESTler has two types of bugs that it can detect. Firstly, RESTler can detect "Error code," a numeric value that tells the system what went wrong. It may also as-

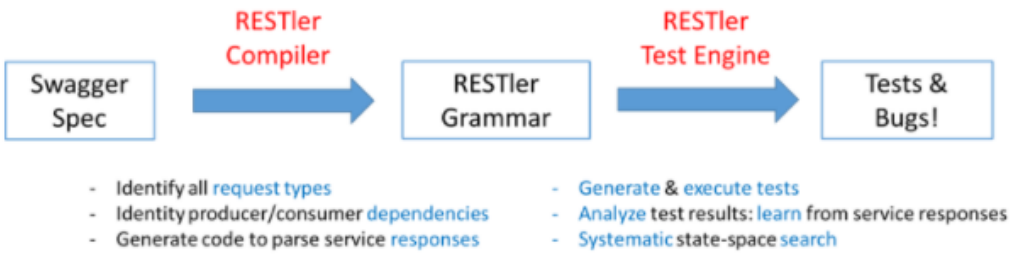


Figure 2.29: RESTler process [13]

sist in the search for a solution to the issue. For example, if a status code 500 ("Internal Server Error") is obtained, this means that there is suspicious input within the system that the API didn't implement to handle. Secondly, "Checker" is utilized for bug finding that can't be done with standard fuzzing. The checker is executed automatically when particular request sequences have been triggered during fuzzing runs. Hence, when RESTler can detect either type of bug or both simultaneously, the Restler will replay the log for triaged issues and use it to recreate the bug.

According to Microsoft and Columbia University's "RESTler: Stateful REST API Fuzzing" publication [79], the limitation of RESTler was discovered during their research. The paper reported that there were several limitations to RESTler. For instance, the requests to API endpoints that contain server-side redirection are not yet handled by RESTler. In addition, procedures involving web-UI interactions aren't supported by RESTler as well. Also, it is reported in the paper [79] that RESTler still can't cover all the vulnerabilities. The vulnerabilities that are not apparent via HTTP status codes (e.g., "Information Exposure") still cannot be detected by such a fundamental test oracle.

Additionally, the team discovered that RESTler still has restrictions on the software's backend. RESTler does not investigate the region of vulnerability in the system's backend. To demonstrate, in certain instances, RESTler can report that the error code: 500 response was discovered but has no idea how it affects the software's backend system. The developer will still need additional network monitoring tools to cover all possible scenarios. As a result, the device is insufficiently efficient to cover all vulnerabilities on both the front and back ends.

In conclusion, RESTler is a Stateful REST API Fuzzer, which implies that it executes request sequences. RESTler can create requests by reading API-specific files

(API Specification). As a result, the tool will know the sequence in which requests are delivered when using RESTler's API Request generation. Consequently, customers may specify criteria for RESTler to enable greater flexibility, such as adding test conditions or an input data collection or writing a new test rule on their software. Thus, there are no limitations on the kind of technology developers may employ to test their services.

2.3 Literature Review

The literature review section provides an overview of project-related research papers and state-of-the-art tools.

2.3.1 Fuzzing Tools and Techniques

Burp suite

Burp Suite [82] a tool used for finding the web application's security vulnerability. Burp can automatically navigate the vulnerability by crawling through the web application. Burp scanners support many of the vulnerabilities caused by the CSRF, stateful functionality, etc. For the API security, Burp can automatically parse the OpenAPI v3 specification written in JSON format to help Burp to identify the API endpoints for testing [83].

When Burp knows the endpoint, it will automatically build the API request and start testing. Burp can find many vulnerabilities related to the APIs, such as SQL injection, XXE attacks. Users are also allowed to test the injection by manually sending injection API requests through Burp, and it will automatically identify if that API is vulnerable to the injection or not. Users can also customize Burp to intercept only for the specific vulnerability type, which will help users categorize for only the vulnerability they are interested in.

Sulley

Sully [84] a fuzzing framework for generating the data input by adopting the fuzzing technique. Sulley contains five major components. The first component is Data Generation. Data Generation will build requests from the primitive and other extended frameworks. The second component is Session Management. After Sully builds the requests, the Session Management will group requests and chain them together in a graph

to form a session. The third component is Agents, which are the interface used for logging and instrumentation such as VMControl, Netmon, Procmon. Then the driver, which is the fourth component, will link testing targets, agents, and requests together. Then the driver will start fuzzing. The last component is Utilities which is a command-line for performing tasks. Sully is the automation testing framework and supports parallel fuzzing, which will help to increase the testing speed and reduce the tedious testing task. To conclude, Sully is an automated fuzzing framework generally used for data generation and monitoring of the test result [85]

BooFuzz

BooFuzz [86] is the improved version of the Sully fuzzing framework that we discussed above. The operational flow of BooFuzz is the same as Sully in that it is still able to generate the data for fuzzing and record the result, but BooFuzz aims to improve network protocol fuzzing. BooFuzz supports serial fuzzing, ethernet and IP layer, and UDP broadcast fuzzing. BooFuzz can also provide better test results with more consistency with CSV export, while Sully can not.

AppSpider

AppSpider [87] the web application security testing tool that can apply to work with SDLC. AppSpider can integrate with CI tools such as Jenkin to help find the security issue at the beginning of the software development phase. Appspider supports up to 95 attack types that cover OWASP's top ten security issues, such as Anonymous Access and Cross-site scripting [88].

AppSpider has one main component called Universal translator. The universal translator will be the one who bridges various testing methods for identifying attacks. The first method that Universal translator uses is web crawling. AppSpider can crawl the running web application to test security issues and does not require source code. This method is perfect for web technology such as HTML and JavaScript, but their APIs are not covered. AppSpider solves the coverage for APIs testing by using the second method to parse Swagger API definition. The third method is recording. AppSpider supports traffic recording passing such as proxy server logs to identify the possible attacking method. The last method is macros; AppSpider will read the interaction behavior of an

application from the recorded interaction to identify that might be possible for testing. The Universal translator will consume information from these sources and then generate the security testing with all these four methods. After the testing finish, App spider will cause the test result that can display to CI/CD tools if integrated

Qualys WAS

Qualys WAS [89] is a cloud-based integration of web application scanning tools and web application firewall. WAS is a cloud-based service, so it is easy to deploy, manage and scale the service. WAS web application scanning tool is built for finding security vulnerabilities for web applications and APIs. WAS use crawling method to scan for vulnerabilities of web application and use Swagger specification phasing for scanning APIs. In comparison, the firewall is used for finding, blocking, and categorizing approved or unapproved web applications in the user network according to the rules that can be customizable. For the result, WAS will visualize the scanning summary in the form of an interactive report for more understanding. WAS also supports CI/CD integration by providing a Jenkins plugin for Jenkins integration [90]

Specification-Based Fuzzing Tools

Other tools that read Swagger specifications in order to parse HTTP requests and guide their fuzzing include the following.

TnT-Fuzzer [91] is an API fuzzing tool that reads swagger specifications (only JSON format) for REST API testing. TnT-Fuzzer will fuzz requests to target API according to the API spec, and it will log the request and response from API. Users can see the crash from this log to identify the API error. Users can use this tool for fuzz custom input requests for testing, such as penetration testing.

APIFuzzer [92] is another API fuzzing tool that can parse API definitions for building API requests. APIFuzzer supports both Swagger specification and OpenAPI specification in JSON and YAML format. Users can fuzz the request body, query string, parameter, and request header through APIFuzzer. HTTP Basic auth is also supported for configuration. Users can inspect the fuzzed result from a test report that APIFuzzer generated. This test report can be generated into JUnit XML format for further CI integration.

2.3.2 Feedback-Directed Test Generation

Random creation of object-oriented software unit tests is the focus of our research. An approach to testing includes feedback from inputs as they are executed, increasing randomness by selecting a method call at random, and placing arguments among the previously constructed inputs. In this technique, inputs are built gradually by choosing a method call and locating arguments from previously generated inputs at random. A collection of contracts and filters is tested against the set of contracts and filters as soon as they are created. Hence, the test's outcome will indicate whether it was a success or a failure. Passing tests is an excellent way to verify that code contracts are maintained while a program develops, while failed checks indicate possible mistakes that need to be rectified (by contravening at least one contract). The experiment highlights the point that many previously undiscovered problems are discovered when feedback-directed random test generation is applied to 14 commonly used libraries totaling 780 KLOC. Additionally, there are less duplicated tests in feedback-directed random test creation as compared to both systematic and undirected random test generation.

2.3.3 General-Purpose Grammar-based Fuzzers

Peach

In 2020, DevOps platform GitLab acquired Peach Tech [93], a security software company specialized in protocol fuzz testing. Peace fuzzing is a cross-platform fuzzer. With Peace, fault detection, data gathering, and automation of the fuzzing environment are all possible using Peach's monitoring system. The customization and expansion of Peach's features are both straightforward. The Peach Fuzzer supports three operating systems, including Windows, MacOS, and Linux. Peach Fuzzer features also cover a wide range of topics, such as mutation algorithms, data types, I/O adapters, monitoring modules, etc. Peace fuzzer can cover lots of features such as Smart & Dumb fuzzing, debuggers, network capture, VM control, GUI Validation Tool, and more. One of the things that made Peace very well-known enough to be purchased by Gitlab is that Peace fuzzer tools can focus on many different customer aspects. By focusing on many types of customers, Peach is able to serve its current and future customers better. This in-

cludes internet browsers and network services, mobile devices, industrial control systems (SCADA), and more.

Additionally, Peach is often considered the finest tool for API and file format testing. Now, with the integration of Peach Tech technologies into GitLab's roadmap for application security testing. Developers will have a simpler time discovering, repairing, and resolving security problems. Additionally, these additional choices are made available inside the GitLab CI/CD environment, making the GitLab DevSecOps offer the first security solution to support both coverage-guided and behavioral fuzz testing and the first true DevSecOps platform to shift fuzz test left.

SPIKE

SPIKE [94] is an open-source set of collections of C API to create fuzzer strings that target network-based protocol. Developers can write scripts in a .spk file and use a scripting language to automate fuzzing tests. SPIKE also comes with a network utility function that can be connected to test more efficiently.

Autodafé

Autodafé [95] fuzzer with block-based grammar to create fuzzing strings like SPIKE but paired with a UNIX debugger to weight in input that uses unsafe functions to be fuzz first.

2.3.4 Whitebox Fuzzing

In the study by Godefroid et al. [96], The authors used compiled x86 code as "whitebox" and fuzzing the program using a combination of symbolic execution, which is trying to find the input of the program by following the performance and reversing operation done to the input variable, and generation search which searching by giving information that leads to new path a priority. Additionally, it was suggested that the multitude of programming languages and even platform architecture make this technique hard to follow.

2.3.5 WebGoat

WebGoat [97] is an open-source software that was designed with the goal of discovering vulnerabilities in Java-based applications that depend on publicly available open-source components. WebGoat enables interested developers to analyze Java-based applications for common security flaws. Web goat enables developers to acquire knowledge via explanation, practice, and mitigation. To demonstrate their understanding of a security issue, students must exploit a real-world vulnerability in the WebGoat application. To steal bogus credit card numbers, for instance, the user needs to utilize SQL injection. There are ideas and code to assist users in a more realistic educational environment provided by the program

2.3.6 Damn Vulnerable Web Application (DVWA)

DVWA [98] is an abbreviation for Damn Vulnerable Web App and is a PHP/ MySQL web application. Its primary goal is to allow security professionals to assess their skills and tools in a legal context, to assist web developers in better understanding web application security processes, and to assist teachers/students in teaching/learning web application security in a classroom environment.

2.4 Chapter Summary

This chapter discusses the project's fundamental concepts, such as web APIs, monolithic applications, microservices applications, BFF (Backend-to-Frontend) communication, OWASP Top 10 API Security (API 3:2019 — Excessive data exposure), API testing, and API fuzzing. Additionally, this chapter explored tools and technologies such as Zeek for network monitoring, RESTler for fuzzing tool development, and penetration test tool development. Finally, this chapter conducts a literature assessment of currently available fuzzing tools and technologies, concluding that there are no currently available tools capable of effectively covering and detecting vulnerabilities and excessive data exposure concerns. As a result, it was determined that the most effective process for mitigating all vulnerabilities is to build an effective API security strategy for the BFF design pattern for microservices. Thus,"Microusity" is constructed with this objective in mind.

CHAPTER 3
ANALYSIS AND DESIGN

The Analysis and Design chapter discusses the key concepts used in the project, and illustrates the system design of the project from the overview to the detailed steps in each process.

3.1 Microusity: Security Testing Tool for Backend for Frontend (BFF) Microservices

To address the defined problem statements and to accomplish the project's goals. We propose an automated tool called "Microusity" (Mi-cro-u-si-ty) to tackle the problems. "Microusity" is an API security testing framework specifically developed for the Backend-to-Frontend (BFF) architectural pattern. The framework is implemented as a web application. Microusity enables the companies, developers, and interested individuals to inspect their microservice systems for potential API vulnerabilities that may result in catastrophic issues such as leaking of sensitive customer data. The service of Microusity includes API fuzzing to locate the API's security vulnerabilities. This is done by automatically analyzing the API specifications and generating the unique fuzzing strings as API test values. Moreover, Microusity includes a network monitor behind the BFF to trace the requests that are further propagated into the back-end systems. This allows Microusity to identify exactly the path of an API security issue from the beginning to the end. This include the original request from client to BFF, the sub-requests BFF creates to the back-end system, and the response from BFF back to client. With Microusity, the developers can find the system's potential security breaches test results can be seen in the tool's test report. Additionally, Microusity's data visualization component enables the understanding of test results and findings. By giving a graphical data visualization to represent the system's API security overview, Microusity helps the developers to be able to comprehend the detected security issues easily. Furthermore, The Microusity system is designed to educate and raise awareness to the developers about the detected

vulnerabilities such as excessive data exposure (i.e., OWASP top 10 API security - API 3:2019 Excessive Data Exposure [12]).

Microusity detection tools can cover three scenarios. In the first scenario (Figure 3.1), the BFF and the core API both perform well. Meaning that our devices will be about to report that The core API responds with no indication of any leaking problems, and the BFF may remove the details of leakage errors from the Core API. On the other hand, the BFF can cover the second scenario (Figure 3.2) when the BFF is good. Still, the core API is terrible, and therefore in this situation, the Microusity system will determine that the Core API responded with details about the leaking fault. Lastly, the Microusity detection tool can detect the third scenario (Figure 3.3), which is when both BFF and core API are bad, which. For this case, Microusity tool will report that the BFF provides comprehensive information about leakage errors from the core API, and the Core API returns extensive information about leakage errors.

Figure 3.1: Good BFF and Good Core API

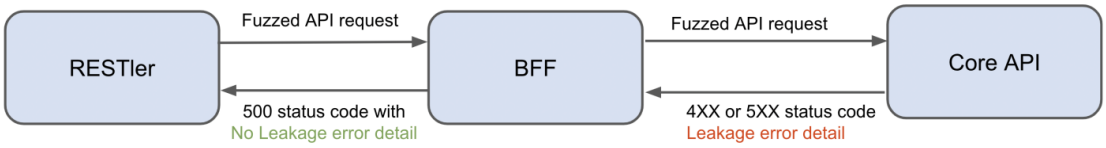


Figure 3.2: Good BFF and Bad Core API

Figure 3.3: Bad BFF and Bad Core API

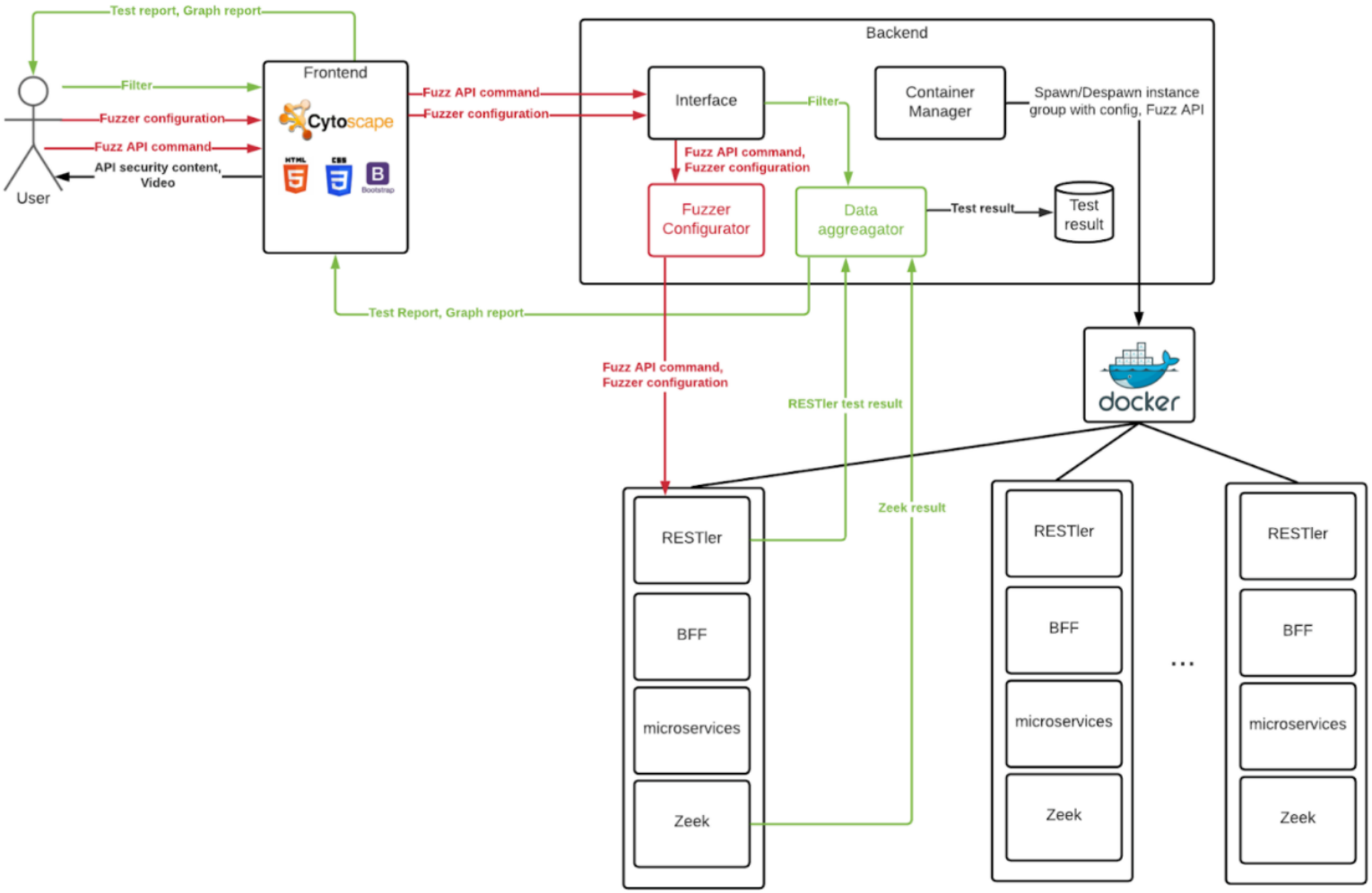


Figure 3.4: Microusity System Architecture

3.2 System Architecture Overview

The system architecture is depicted in Figure 3.4. For the frontend, The users (i.e., developers) give the Fuzzer configuration, fuzz API commands, and filters to the Microusity tool via its frontend. The test report and graph from the testing will be visualized by Cytoscape.js, which is a framework that can create the graph for visualization of the API request and response sequences by giving the JSON file format to make it generate a graph. Cytoscape.js can visualize node and edge graphs that represent the structure of request flow from the test, and it suits best for the visualizations we need in our tool.

The backend is composed of four parts.

1. The first is the **interface**, which handles the data between the frontend and backend. It also queries test results and current test status.

2. The second is the **container manager**, which starts the reproducible test environment, isolates the testing session, and collects the artifact results from the test. It also informs the user if any error occurred between creating instances.

3. The third is the **fuzzer configurator**. It incorporates the custom settings from the interface that users may create to improve the test coverage and test input for finding more bugs.

4. Lastly, the **data aggregator** consumes the data collected from test results and generates the graph between the results of RESTler and Zeek. Then it stores the result in the data storage.

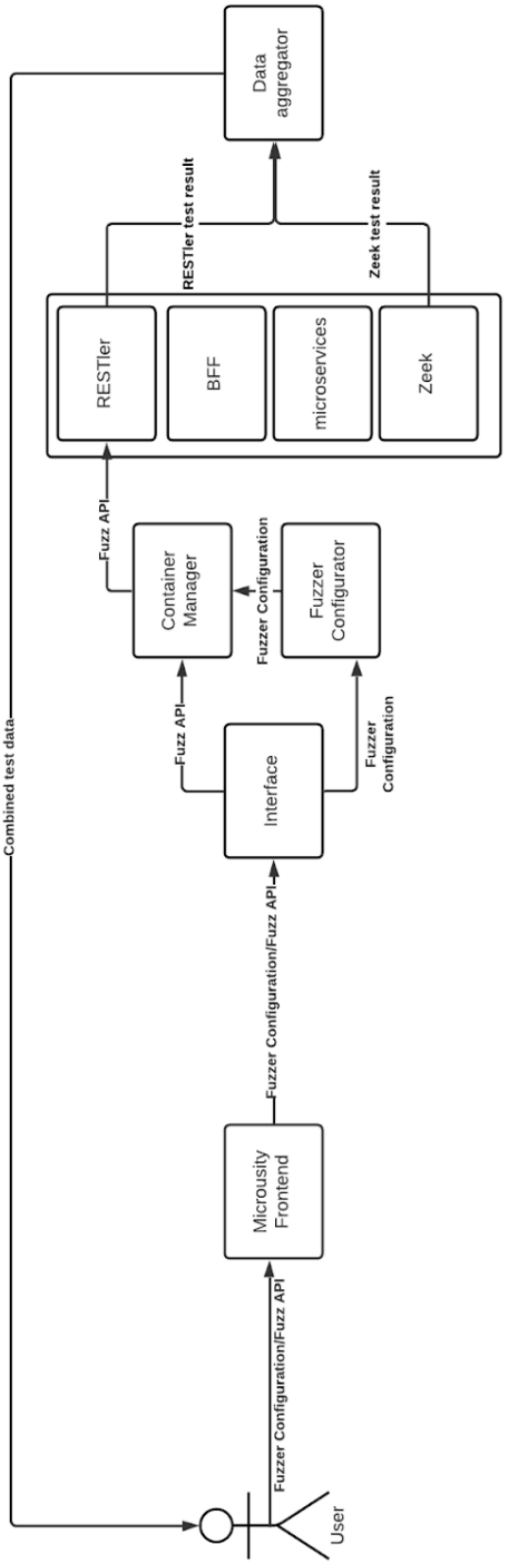


Figure 3.5: Microusity workflow

3.2.1 Workflow of Microusity

The Figure 3.5 shows the workflow of Microusity that consists of:

1. The fuzzer configuration is supplied by the user through the interface and passed to the fuzzer configurator.
2. The container manager initializes the instance group of the container with the supplied fuzzer configuration from the fuzzer configurator.
3. After all containers are initialized, the fuzzer will be started to fuzz APIs.
4. After the fuzzing is completed, the data aggregator will collect the artifact from the test and generate a test report and graph report.
5. The instance will be shut down and torn down.
6. The test report is displayed to the user.

3.2.2 Security Report

The test report (Figure 3.6) details the coverage of the API route that RESTler can successfully access against the Swagger specification's pathways. Additionally, the report highlights HTTP response faults and contains the issue discovered by Microusity. Each error includes the request from BFF, including the request to microservices, which is the source of the issue.

The visualization graph (Figure 3.7) represents the connection that Microusity found during the APIs fuzzing. The left is the connection summary, while the right is what request BFF made after receiving the request from RESTler.

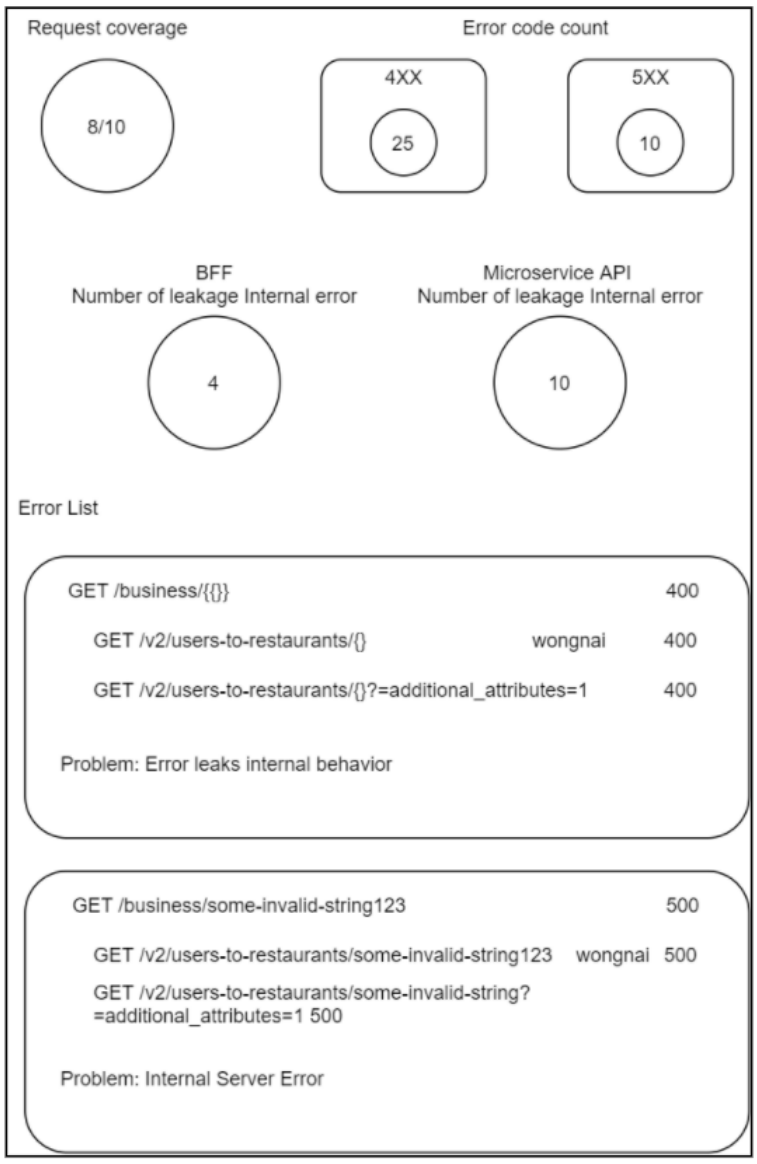


Figure 3.6: Test report: - Overall coverage, Response code and problem detected.

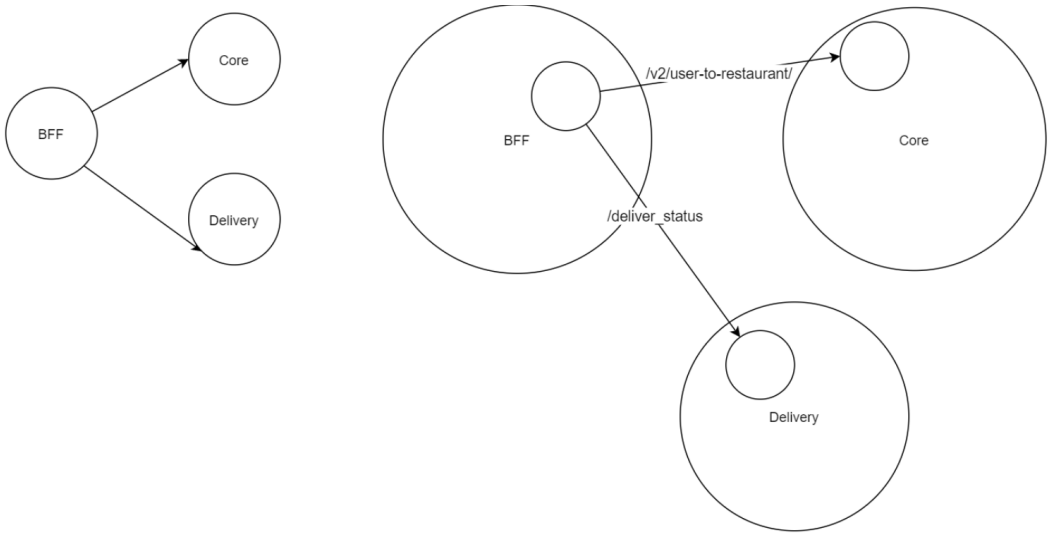


Figure 3.7: Visualization Graph

3.3 Use Case Analysis

In the use case diagram (Figure 3.8), Microusity interacts with one external actor, which is the user (microservice developers). The user interacts with the system that contains several use cases. First, the user can **fuzz the API** to test the security vulnerability of the APIs. The user can **customize custom configuration** in order to add their desired test input in addition from using the default fuzzing input. Then the Microusity system will generate the API fuzzing request for testing. When the API fuzzing is finished, the user can **see the test report**, which contains the details of vulnerabilities that the Microusity system found. For this case, the user can **search for the API request** to see only the sequence of requests that match the search query. Moreover, the user can apply a response status code filter to filter out the unwanted sequences that do not match the filter value. The user can also see the graph report in order to see more detail on which API request and response paths caused the error. In addition, the user can **apply a vulnerability filter** to see only the graph path that contains the specific vulnerability. For the education functionality, the user can **watch the video lesson** and **read API security content** that will enrich knowledge about the API security vulnerability.

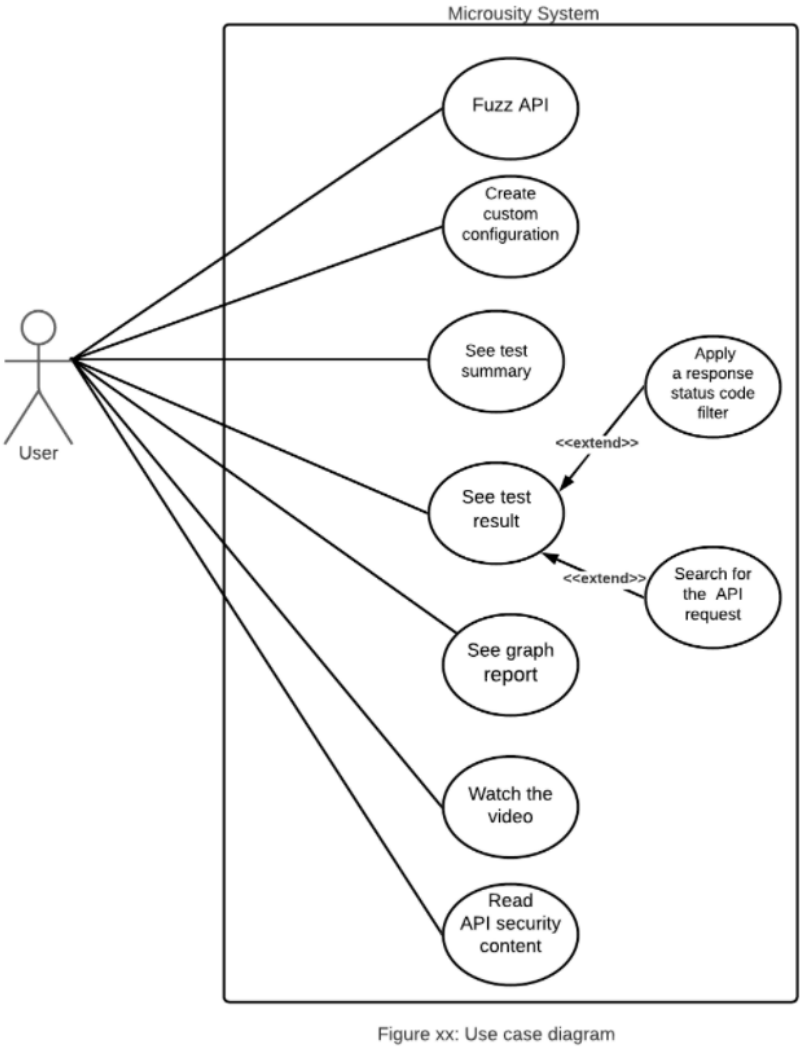


Figure 3.8: Use cases of Microusity

3.4 System Structure

The Microusity system is divided into six modules which are presented in the structure chart as Figure 3.9. These six modules contain Fuzz API, Create custom fuzzing string, See test report, See graph report, Watch the video, Read API security content..

1. The first module is Fuzz API. This module is built for the user to generate the API fuzzing request for API security testing. This module contains two submodules:

   (a) Fuzz lean method: This is a submodule for generating API fuzz testing, which aims to execute quickly.

(b) Full fuzzing method: This is a submodule for generating API fuzz testing, aiming to find more bugs with a longer execution time.

2. The second module is Create custom configuration.

   (a) Create custom fuzz input: This submodule is for the user to put the custom input to make the system generate the fuzzing request according to the custom input.

   (b) Create custom invalid object input: The user can check more vulnerabilities by adding more invalid object input through this submodule.

3. The third module is See test report. This module will generate the test report according to the test result. The user can see details of test results with the testing summary. This module contains two submodules.

   (a) Apply response status code filter: This submodule is built for filtering to show the report only the selected response status code.

   (b) Search for the API request: The user can see only desired API requests by searching the API request through this submodule.

4. The fourth module is See graph report. The user can see test results with more detail in the form of a graph. With this module, users can see a clearer picture of which API request sequence leads to security vulnerability.

5. The fifth module is Watch the video. This module is created for the user to watch the video about API security vulnerability and API security testing.

6. The last module is Read API security content. This module is for the user to read the static article and content about API security vulnerability and API security testing.

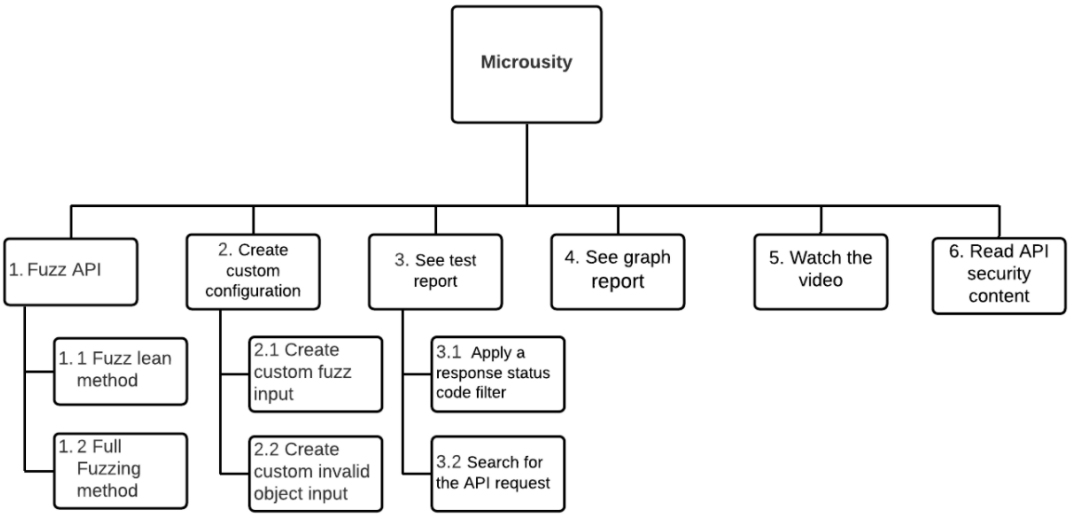


Figure 3.9: Microusity Structure Chart

3.5 System Analysis

3.5.1 Data Flow Diagram Level 0 (Context Diagram)

Figure 3.10 shows the data flow between the user and the Microusity System. The user provides the fuzzer configuration input and the Fuzz API command to the system, and the system will use these inputs to fuzz APIs and pass the test report back to the user. The user can input the search query and apply the report filter to the Microusity system. Then the system will use this query to display the test result that matches the query and generate a graph report for the user to see the test result in graph format. The test result's history will be kept in the system for the user to see the previous reports. The user can select the video to watch API security contents or select the API security contents from the system.

3.5.2 Data Flow Diagram Level 1

Figure 3.11 shows the Data flow in the Microusity system and the user. The Microusity system is divided into six subprocesses. Those are Fuzz API, Create custom fuzz string, See test report, See graph report, Watch the video, and lastly, Read API security content. Subprocess 3 is the one who interacts with the Test Result data storage to keep the test history for the user to retrieve later on. Subprocess 4 will use the manipulated test result that was filtered and queried for only the wanted APIs sequence by

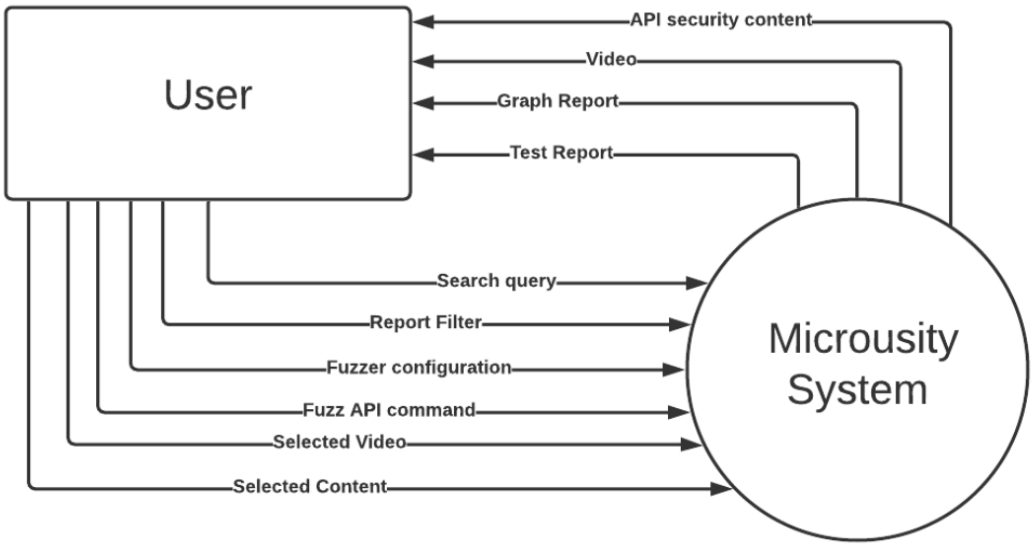


Figure 3.10: Data Flow Diagram Level 0

subprocess 3 to display the graph report for the user. This manipulated test result will not keep in the Test Result storage.

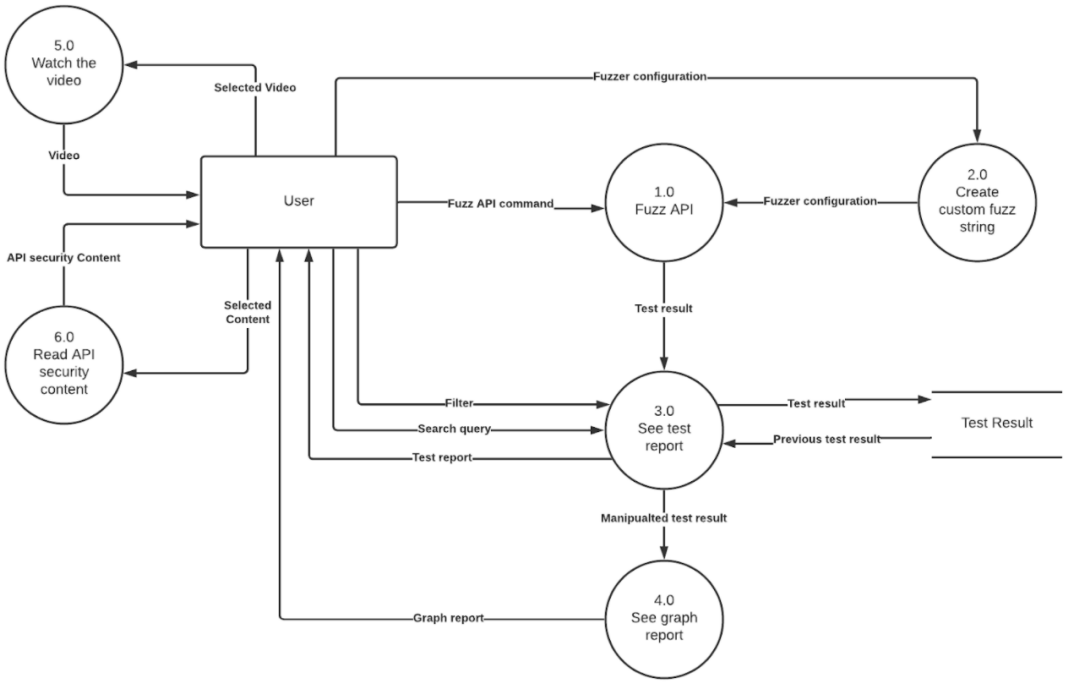


Figure 3.11: Data Flow Diagram Level 1

3.6 Interface design

As the primary component of Microusity is the tool for Backend for Frontend (BFF) Microservices and educational modules, users will be able to test and learn by using the Microusity sandbox system. The Figure 3.12 shows the overall interface of Microusity and Figure 3.13 shows overview of the detection tool and education aspects.

The user may Fuzz the API on this page (Figure 3.14) and then generate a custom fuzzing string to test for issues of excessive data exposure. Following that, this module will create a test report based on the test outcome. With the testing summary, the user may see detailed information about the test outcomes easily through our visualizer.

As for the educational part (Figure 3.15), students will be able to learn by doing and learning from the sandbox system that simulates various problems (OWASP Top Ten Security) so that students can learn about system monitoring and system protection. In the first part, when students enter the Sandbox Simulator page, they will learn about different types of problems. How to check various problems and detailed information about them When the lesson is complete, there will be a quiz during the class to do.

The homepage of Microusity (Figure 3.16) includes an introduction to the Mi-

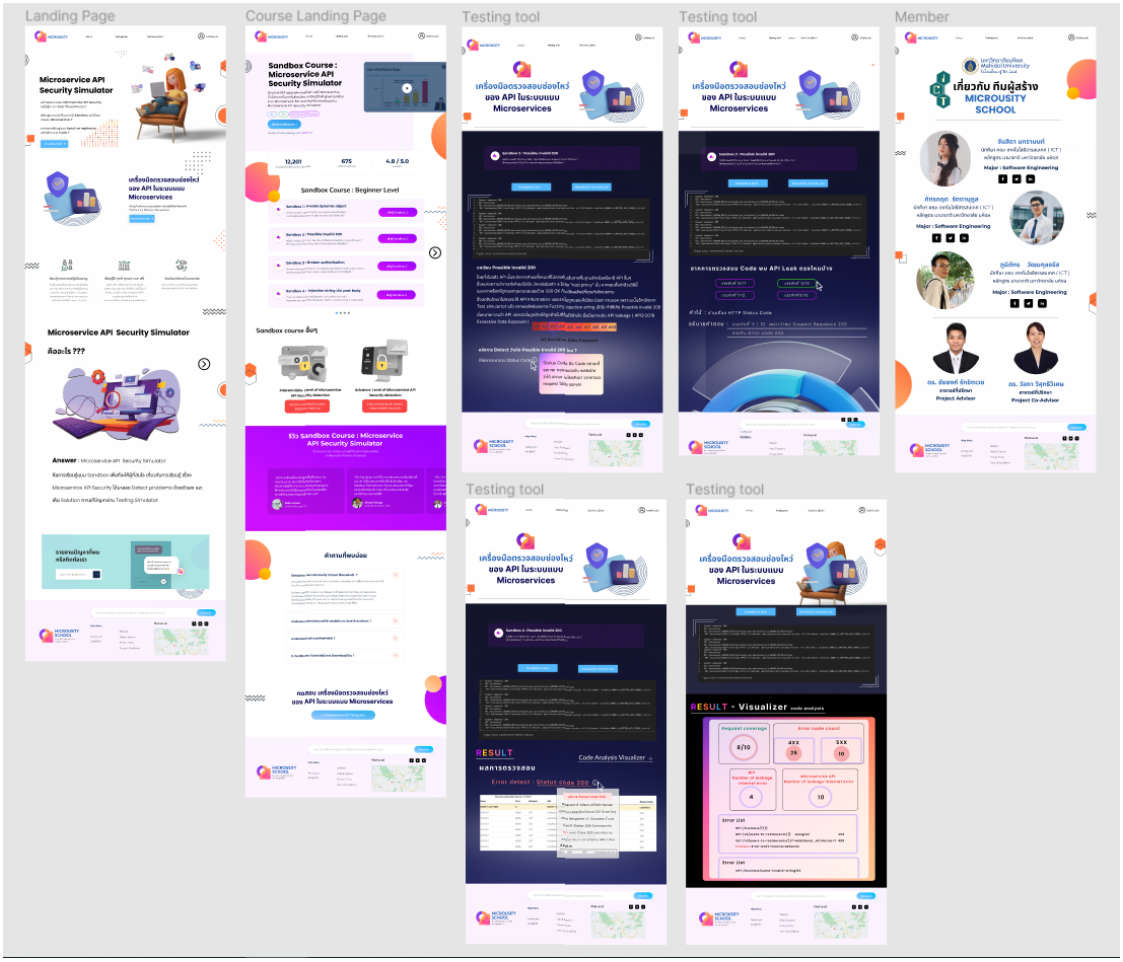


Figure 3.12: Overview of the interface design of Microusity

crousity web application. By introducing Microusity the detection tool, an introduction to the educational part, the advantages of studying with Microusity Sandbox, and more

On the course page (Figure 3.17), there are several elements. For instance, a course review section from real students so that interested parties can read the reviews and other opinions of other students before choosing to study, or a short video introduction. There is also an FAQ section to provide information to students and those interested in learning about the Sandbox course.

Lastly, the team page (Figure 3.18) presents the team members who contributed to making this project come true, as well as the professor who help guidance us through the entire process.

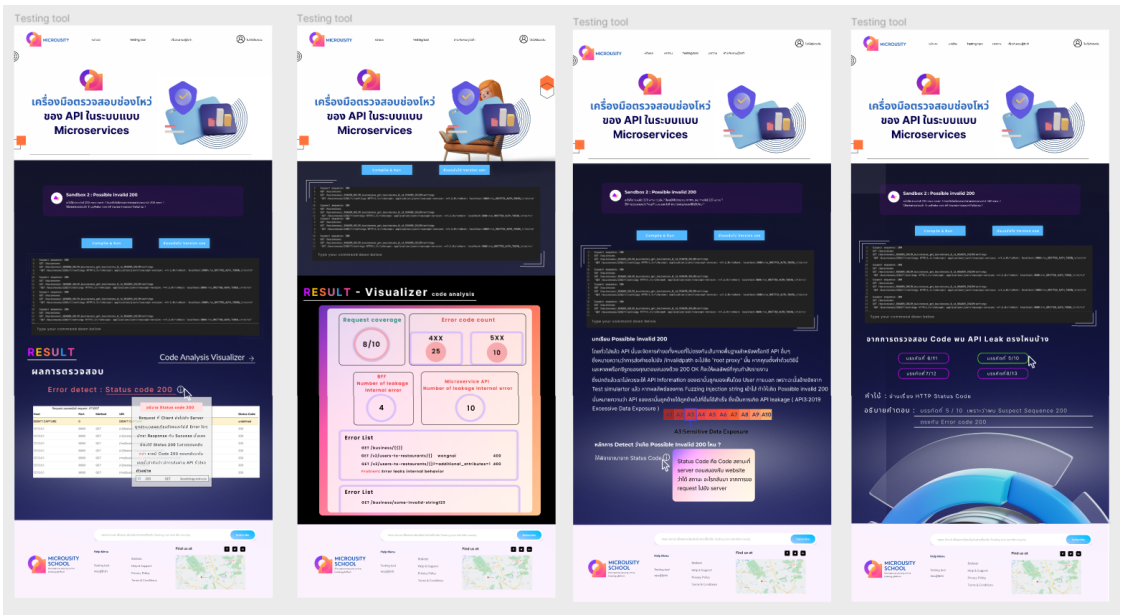


Figure 3.13: Overview the detection tool and education aspects

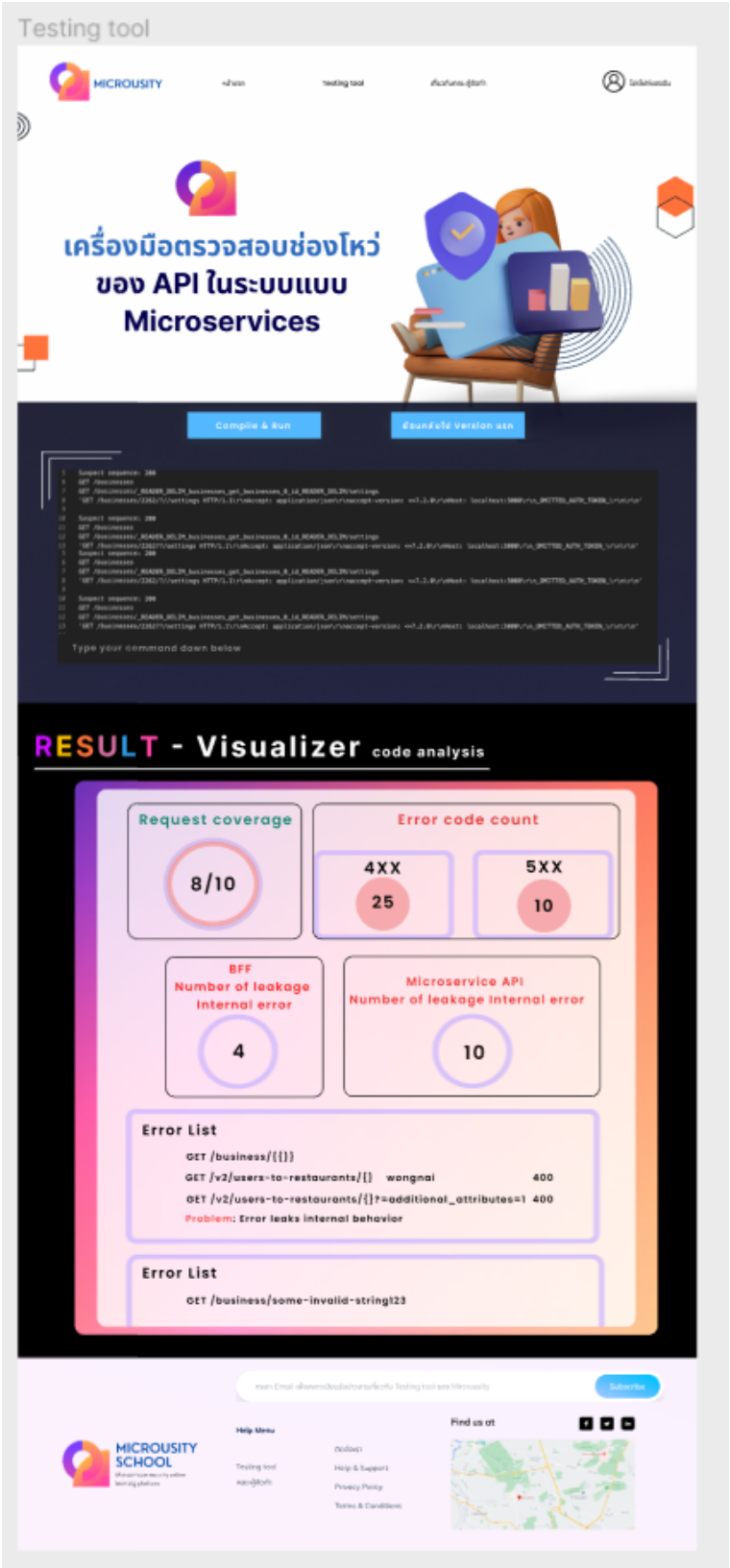


Figure 3.14: Microusity detection tool for Backend for Frontend (BFF) Microservices

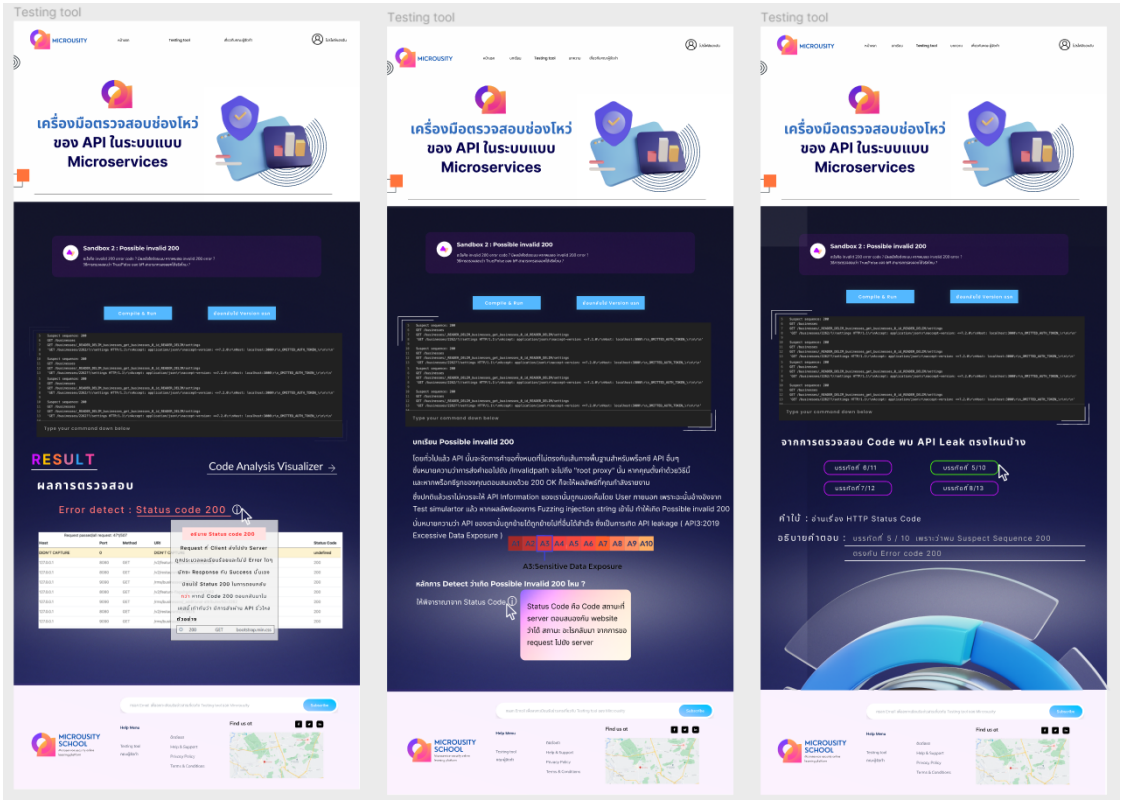


Figure 3.15: Microusity educational part for learner

Figure 3.16: Microusity home page

Figure 3.17: Microusity course page

Figure 3.18: Microusity team members page

3.7 Comparison to Related Work

Table 3.1 below shows the comparison of features from each API security testing tool. In the Literature Review section, the discussion of how each testing tool works were mentioned. This comparison section will discuss how different the Microusity system is compared with other existing tools. The Burp Suite and AppSpider are picked for the comparison because these API security testing tools are widely used and used as commercial tools. For the fuzzing framework, the BooFuzz is the representation of the standard fuzzing framework.

There are several features needed to be compared. First, the API specification phasing, Burp Suite, AppSpider, and Microusity contain this feature to identify the testing target and generate the API testing request, but BooFuzz does not include this feature. Second, web crawling, both Burp Suite and AppSpider can crawl through web applications to identify the test, but BooFuzz and Microusity cannot crawl the web page to generate the test. Third, Burp Suite and Microusity can generate the stateful API testing, but for the Burp Suite, the users have to manually look through the test result by themself to identify the State of each API request and respond while Microusity can automatically do it. Fourth, all tools support the API fuzzing technique except the AppSpider. Fifth, only the Microusity can support BFF and microservices' error traceability, while the other tools can support only the front side. Sixth, for the readable report interface, only BooFuzz does not contain this feature. BooFuzz provides only raw static text files and logs as a result of the testing, and this leads to a complicated interpretation for the user to understand the test result. Seventh, for the graph report visualization, Microucity can report the test result into the graph format for helping the users understand the API request and response sequence caused the scalability while other tools only support the text-based report. Lastly, only Microusity contained API security content for helping the user to understand the key point of API security vulnerability.

To conclude, from the comparison above, Microusity can cover all of the necessary features for API security vulnerability testing and is built to be different by providing the graph report for more understandability when identifying the error. Moreover, Microusity is created especially for the BFF microservice architecture pattern to trace the

Table 3.1: Comparison of Microusity to Similar Tools

Features	Burp Suite	AppSpider	BooFuzz	Microusity
Read API specification	x	x		x
Web crawling	x	x		
Stateful REST API testing	x			x
API Fuzzing technique	x		x	x
Allow custom input	x	x	x	x
Trace Error between BFF and microservices				x
Readable report interface	x	x		x
Graph report visualization				x
Educational content				x

error from BFF more efficiently than other tools.

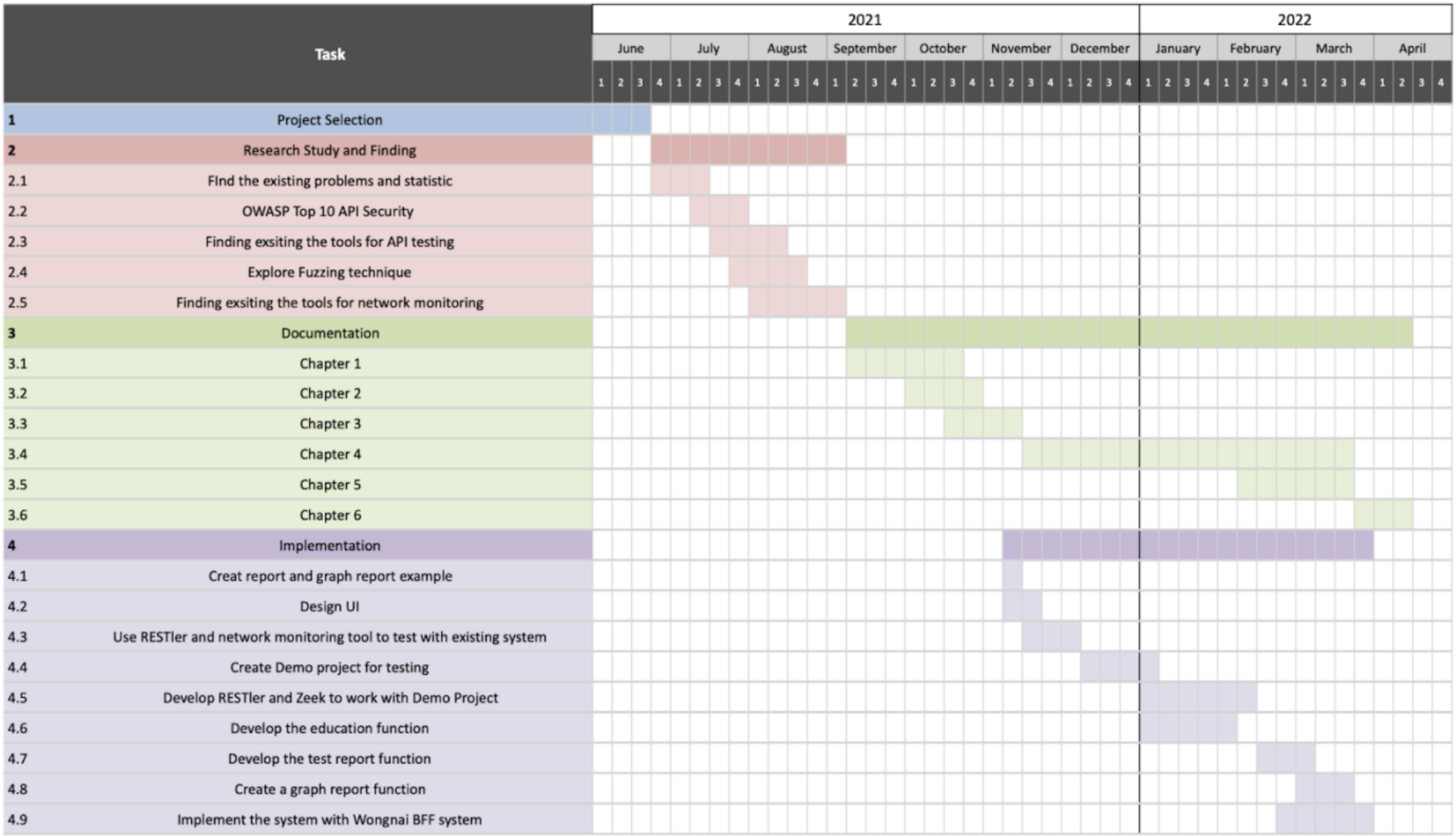


Figure 3.19: Project timeline

3.8 Project Timeline, Current Progress, and Future Work

3.8.1 Project Timeline

Figure 3.19 shows the project timeline of Microusity. The project selection phase occurred in June, then the research for project background and essential information for the project was conducted from July to September. The documentation phases began in September after the finding phase ended and lasted until April 2022. The implementation phase started with the UI design and report design that began in November. As soon as the idea is solid and validated by the committee after the proposal presentation in November, the Microusity will be developed and implemented until March 2022.

3.9 Chapter Summary

This chapter discusses the study and design of the Microusity system, which is a proposed solution to solve an issue of API 3:2019 Excessive Data Exposure [12] within the Microservices architecture style. Therefore, Microusity is an application that performs API security testing on BFF systems. With the ability to also create visualizations to help security testers understand the security issues easily, through the use of Microusity report. As well as offering educational content related to the detected API security issues. In order to increase the developer's awareness and expertise of these issues. Additionally, this chapter includes an introduction to Microusity, the system architecture, the use case diagram, the structure chart, and the level 0 and level 1 data flow diagrams. Additionally, this chapter discusses the user interface design and provides a comparison to comparable work. Finally, this chapter presents an analysis report utilizing a Gantt chart to illustrate the project's timeline, current work, and future work.

REFERENCES

[1] Kamaruzzaman M.. "Looking beyond the hype: Is modular monolithic software architecture really dead?", Towards Data Science; Dec 2020, [Online]. Available: https:// towardsdatascience.com/ looking-beyond-the-hype-is-modular-monolithic-software-architecture-really-dead-e386191610f8.

[2] McCracken H.. "Fifty Years of Basic, the language that made computers personal", Time; Apr 2014, [Online]. Available: https://time.com/69316/basic/.

[3] Gültekin M.. "Microservices and Microservice Architecture", Medium; Aug 2021, [Online]. Available: https:// medium.com/ @mertgltekin_58750/ microservices-and-microservice-architecture-3fa69dba089b.

[4] Babu S.. "8 core components of Microservice architecture"; Sep 2021, [Online]. Available: https://www.optisolbusiness.com/insight/8-core-components-of-microservice-architecture.

[5] Abbasi A.. "Message communication patterns for application integration"; Aug 2021, [Online]. Available: https:// tutorialspedia.com/ understanding-message-communication-patterns-for-application-integration/.

[6] Amazoncasestudy. "Alt/S Case Study", Amazon Web Services, Inc; 2016, [Online]. Available: https://aws.amazon.com/solutions/case-studies/alts/.

[7] Cockcroft A.. "The state of the art in microservices by Adrian Cockcroft", YouTube; Jan 2015, [Online]. Available: https:// www.youtube.com/ watch?v=pwpxq9-uw_0.

[8] Kappagantula S.. "Microservice architecture - learn, build, and deploy applications - dzone microservices"; Jul 2018, [Online]. Available: https://dzone.com/articles/microservice-architecture-learn-build-and-deploy-a.

[9] Elias JM.. “Backend for frontend (BFF) pattern”, Zinklar Tech; Nov 2019, [Online]. Available: https://medium.com/zinklar-tech/backend-for-frontend-bff-pattern-5e8810779d9f.

[10] Newman S.. “Sam Newman - backends for frontends”; Nov 2015, [Online]. Available: https://samnewman.io/patterns/architectural/bff/.

[11] SoundCloud. “Service architecture at soundcloud - part 1: Backends for frontends”;, [Online]. Available: https://developers.soundcloud.com/blog/service-architecture-1.

[12] Isbitski2 M.. “API3:2019: Excessive data exposure”; Feb 2021, [Online]. Available: https://salt.security/blog/api3-2019-excessive-data-exposure.

[13] Microsoft. “Microsoft/Restler-Fuzzer: Restler”;, [Online]. Available: https://github.com/microsoft/restler-fuzzer.

[14] Bookutils. “Breaking things with random inputs - the fuzzing book”;, [Online]. Available: https://www.fuzzingbook.org/beta/html/Fuzzer.html.

[15] Scheunemann P.. “What is Docker”, Abdo Pub. Co.; 2002, [Online]. Available: https://aws.amazon.com/th/docker/.

[16] ZeekProjectRevision. “About zeek”, The Zeek Project Revision;, [Online]. Available: https://docs.zeek.org/en/v4.0.0/about.html.

[17] Newman S., “Building Microservices”. feb 2015;1, [Online]. Available: https://www.oreilly.com/library/view/building-microservices/9781491950340/.

[18] IBM Cloud Education I.. “Microservices”;, [Online]. Available: https://www.ibm.com/cloud/learn/microservices.

[19] Research and Markets. “Cloud microservices market - growth, trends, COVID-19 impact, and forecasts (2021 - 2026)”;, [Online]. Available: https://www.researchandmarkets.com/reports/4787543/cloud-microservices-market-growth-trends.

[20] Shriramvenugopal. “The story of netflix and microservices”; May 2020, [Online]. Available: https:// www.geeksforgeeks.org/ the-story-of-netflix-and-microservices/.

[21] Siriwardena P.. “Building microservices”, FACILELOGIN; Mar 2018, [Online]. Available: https:// medium.facilelogin.com/ building-microservices-designing-fined-grained-systems-d37b57a97c4e.

[22] Richardson C.. “Microservices pattern: Database per service”;, [Online]. Available: https://microservices.io/patterns/data/database-per-service.html.

[23] SALT LABS SS.. “Highlights from the 2021 state of API security report”;, [Online]. Available: https:// faun.pub/ highlights-from-the-2021-state-of-api-security-report-4b8abb201149?

[24] Isbitski M.. “Details of the owasp API security top 10”, SALT Security; Feb 2020, [Online]. Available: https:// salt.security/ blog/ owasp-api-security-top-10-explained.

[25] Bureau E.. “Crystal-gazing cybersecurity, 2020 - Cyberthreat landscape 2020”; Dec 2019, [Online]. Available: https:// economictimes.indiatimes.com/ industry/ tech/ crystal-gazing-cybersecurity-2020/ api-will-be-exposed-as-the-weakest-link-leading-to-cloud-native-threats/slideshow/72398222.cms.

[26] Ruff J.. “Web apis, Web Services, & microservices: Basics & differences”; Oct 2021, [Online]. Available: https://www.parasoft.com/blog/web-api-vs-web-services-microservices-basics-differences/.

[27] IBM; Dec 2020, [Online]. Available: https://www.ibm.com/docs/en/cics-ts/5.2?topic=services-what-is-web-service.

[28] Monus A.. “Soap vs rest vs JSON - a 2021 comparison”, Raygun Blog; Mar 2021, [Online]. Available: https://raygun.com/blog/soap-vs-rest-vs-json/.

[29] Sustainability of Digital Formats. “Sustainability of digital formats: Planning for Library of Congress Collections”, Library of congress collection; Jan

2014, [Online]. Available: https://www.loc.gov/preservation/digital/formats/fdd/fdd000381.shtml.

[30] Shankar R.. "JSON vs XML in 2021: Comparison, features & example"; Jan 2021, [Online]. Available: https://hackr.io/blog/json-vs-xml.

[31] Truica CO., Radulescu F., Boicea A., Bucur I., "Performance evaluation for CRUD operations in asynchronously replicated document oriented database", 20th International Conference on Control Systems and Computer Science. 2015;[Online]. Available: https://bit.ly/3Hgxjsi.

[32] Nurseitov N., Paulson M., Reynolds R., Izurieta C., Comparison of JSON and XML data interchange formats: A case study. Jan 2019;.

[33] Breje AR., Gyorödi R., Gyorödi C., Zmaranda D., Pecherle G., "Comparative study of data sending methods for XML and JSON models", International Journal of Advanced Computer Science and Applications. 2018;9(12).

[34] Haq Su.. "Introduction to monolithic architecture and microservices architecture", KoderLabs; Jul 2018, [Online]. Available: https://medium.com/koder-labs/ introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63.

[35] Richardson C.. "Microservices pattern: Monolithic Architecture Pattern";, [Online]. Available: https://microservices.io/patterns/monolithic.html.

[36] Francesco PD., Malavolta I., Lago P., "Research on architecting microservices: Trends, Focus, and potential for industrial adoption", 2017 IEEE International Conference on Software Architecture (ICSA). 2017;.

[37] Richardson C.. "Microservices pattern: Circuit breaker";, [Online]. Available: https://microservices.io/patterns/reliability/circuit-breaker.html.

[38] CDC-Info. "Health Insurance Portability and accountability act of 1996 (HIPAA)", U.S. Department of Health & Human Services; Sep 2018, [Online]. Available: https://www.cdc.gov/phlp/publications/topic/hipaa.html.

[39] Consulting I.. “General Data Protection Regulation GDPR”; Sep 2019, [Online]. Available: https://gdpr-info.eu/.

[40] Qian A.. “Advantages and disadvantages of microservices architecture”, Cloud Academy; May 2020, [Online]. Available: https:// cloudacademy.com/ blog/ microservices-architecture-challenge-advantage-drawback/.

[41] Kwiecien A.. “10 companies that paved the way for developing microservices”, Divante; Aug 2019, [Online]. Available: https:// www.divante.com/ blog/ 10-companies-that-implemented-the-microservice-architecture-and-paved-the-way-for-others.

[42] Fulton III SM.. “What led Amazon to its own microservices architecture”, The New Stack; May 2021, [Online]. Available: https://thenewstack.io/led-amazon-microservices-architecture/.

[43] Amazon. “Solutions”, National Council on Vocational Education; 1991, [Online]. Available: https://aws.amazon.com/solutions/case-studies/bridestory/.

[44] Nguyen CD.. “A design analysis of cloud-based microservices architecture at Netflix”, The Startup; May 2020, [Online]. Available: https:// medium.com/ swlh/ a-design-analysis-of-cloud-based-microservices-architecture-at-netflix-98836b2da45f.

[45] Gluck A.. “Introducing domain-oriented Microservice architecture”; Sep 2020, [Online]. Available: https://eng.uber.com/microservice-architecture/.

[46] Villamizar M., Garces O., Castro H., Verano M., Salamanca L., Casallas R., et al., “Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud”, 2015 10th Computing Colombian Conference (10CCC). 2015;.

[47] Kamaruzzaman M.. “Microservice Architecture: A brief overview and why you should use it in your next project”, Towards Data Science; Dec 2020, [Online]. Available: https://towardsdatascience.com/microservice-architecture-a-brief-overview-and-why-you-should-use-it-in-your-next-project-a17b6e19adfd.

[48] Martinelli S., Samani S., Tucker E., Oliver J., Chang J.. “Backend for frontend application architecture”, IBM; Dec 2018, [Online]. Available: https://developer.ibm.com/patterns/create-backend-for-frontend-application-architecture/.

[49] Calçado P.; 2020, [Online]. Available: https://philcalcado.com/.

[50] Creixell J.. “Service architecture at soundcloud - part 1: Backends for frontends”, SoundCloud; Jul 2021, [Online]. Available: https://developers.soundcloud.com/blog/service-architecture-1/.

[51] Microsoft. “Backends for frontends pattern - cloud design patterns”;, [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/patterns/backends-for-frontends.

[52] OWASP. “WHO IS THE OWASP® foundation?”;, [Online]. Available: https://owasp.org/.

[53] OWASP2. “Owasp API Security project”;, [Online]. Available: https://owasp.org/www-project-api-security.

[54] Berde A., Hegde P.. “Exclusive: Flaw left user data of 2 million Bounceshare customers vulnerable to hack”, MoneyControl; Nov 2019, [Online]. Available: https://www.moneycontrol.com/news/technology/exclusive-flaw-left-user-data-of-2-million-bounceshare-customers-vulnerable-to-hack-4629331.html.

[55] Katalon. “10 API testing tips for Beginners (Soap & Rest): Complete guide”; Nov 2021, [Online]. Available: https://www.katalon.com/resources-center/blog/api-testing-tips/.

[56] Besic N.. “The Art of Fuzzing”; Sep 2021, [Online]. Available: https://www.neuralegion.com/blog/fuzzing/.

[57] OWASP3. “Fuzzing”;, [Online]. Available: https://owasp.org/www-community/Fuzzing.

[58] Chen C., Cui B., Ma J., Wu R., Guo J., Liu W., “A systematic review of fuzzing techniques”, Computers & Security. 2018;75:118–137.

[59] Hamilton T.. “Fuzz Testing(fuzzing) tutorial: What is, types, tools & example”; Oct 2021, [Online]. Available: https://www.guru99.com/fuzz-testing.html.

[60] TechTarget. “What is Fuzz Testing (fuzzing)?”, TechTarget; Mar 2010, [Online]. Available: https://searchsecurity.techtarget.com/definition/fuzz-testing.

[61] Cunliffe M., Harrington B., Horky K., Chowdhury M., J K., Awais A., et al.. “A brief history of Node.js”;, [Online]. Available: https://nodejs.dev/learn/a-brief-history-of-nodejs.

[62] Dahl R.. “Ryan Dahl: Node JS”, YouTube; Oct 2012, [Online]. Available: https://www.youtube.com/watch?v=EeYvFl7li9E&t=9s.

[63] Izle S., Stacy. “Node.js advantages: What do you need to know before starting the project?”; Aug 2021, [Online]. Available: https://procoders.tech/blog/advantages-of-using-node-js/.

[64] Paper D.. “Node.js summary”; Apr 2021, [Online]. Available: https://developpaper.com/node-js-summary-2/.

[65] Perry B.. “6 real-world applications of node.js”; Jun 2019, [Online]. Available: https://businessingmag.com/8371/equipping/node-js/.

[66] Ono K.;, [Online]. Available: https://cytoscape.org/what_is_cytoscape.html.

[67] Slenter D.. “Wikipathways”;, [Online]. Available: https://www.wikipathways.org/index.php/WikiPathways.

[68] Reactome. org. “Home”;, [Online]. Available: https://reactome.org/.

[69] Genome. “KEGG PATHWAY Database”; 2021, [Online]. Available: https://www.genome.jp/kegg/pathway.html.

[70] Lang L., Pettkó-Szandtner A., Tunçay Elbaşı H., Takatsuka H., Nomoto Y., Zaki A., et al., “The dream complex represses growth in response to DNA damage in Arabidopsis”, Life Science Alliance. 2021;4(12).

[71] Shannon P., "Cytoscape: A software environment for integrated models of Biomolecular Interaction Networks", Genome Research. 2003;13(11):2498–2504.

[72] Pico A.. "Cytoscape Ecosystem tutorial"; Sep 2021, [Online]. Available: https://cytoscape.org/cytoscape-automation/for-scripters/R/notebooks/Cytoscape-ecosystem-tutorial.nb.html.

[73] Franz M., Lopes CT., Huck G., Dong Y., Sumer O., Bader GD., "Cytoscape.js: A graph theory library for visualisation and analysis", Bioinformatics. 2015;.

[74] Ilievski V.. "JavaScript: Graph visualization using cytoscape JS", Analytics Vidhya; Apr 2020, [Online]. Available: https://medium.com/analytics-vidhya/javascript-graph-visualization-using-cytoscape-js-e741556afb96.

[75] Scheunemann P.. "What is Docker", Abdo Pub. Co.;, [Online]. Available: https://aws.amazon.com/th/docker/.

[76] Syed A., Karthic K.. "Docker & The Rise of microservices";, [Online]. Available: https://timber.io/blog/docker-and-the-rise-of-microservices/.

[77] Wainstein AL., Kovačević AA., Ritz AN., Rajput AM., Pandit AK., Cornelissen AJ., et al.. "Docker use cases - how to handle big data with Docker"; Dec 2018, [Online]. Available: https://bigdata-madesimple.com/docker-use-cases-how-to-handle-big-data-with-docker/.

[78] Godefroid P., Lehmann D., Polishchuk M.. "Differential regression testing for rest apis"; Nov 2020, [Online]. Available: https://www.microsoft.com/en-us/research/publication/differential-regression-testing-for-rest-apis/.

[79] Atlidakis V., Godefroid P., Polishchuk M.. "Restler: Stateful rest api fuzzing"; Mar 2021, [Online]. Available: https://www.microsoft.com/en-us/research/publication/restler-stateful-rest-api-fuzzing/.

[80] Atlidakis V., Godefroid P., Polishchuk M.. "Checking security properties of cloud services rest apis"; Nov 2020, [Online]. Available: https://www.microsoft.com/en-us/research/publication/checking-security-properties-of-cloud-services-rest-apis/.

[81] Microsoft. “Restler-fuzzer/ compiling.md at main · Microsoft/ Restler-Fuzzer”;, [Online]. Available: https://github.com/microsoft/restler-fuzzer/blob/main/docs/user-guide/Compiling.md.

[82] PortSwigger. “BURP suite - application security testing software”;, [Online]. Available: https://portswigger.net/burp.

[83] PortSwigger2. “API security testing software from Portswigger”;, [Online]. Available: https://portswigger.net/burp/vulnerability-scanner/api-security-testing.

[84] Amini P., Ryan. “OpenRCE/sulley: A pure-python fully automated and unattended fuzzing framework.”;, [Online]. Available: https://github.com/OpenRCE/sulley.

[85] Amini P., Portnoy. “Fuzzing sucks ! Introducing Sulley Fuzzing framework”;, [Online]. Available: https://github.com/OpenRCE/sulley/blob/master/docs/introducing_sulley.pdf.

[86] Jtpereyda J.. “JTPEREYDA/Boofuzz: A fork and successor of the Sulley Fuzzing Framework”;, [Online]. Available: https://github.com/jtpereyda/boofuzz.

[87] AppSpider. “Web application security testing with AppSpider”;, [Online]. Available: https://www.rapid7.com/products/appspider/.

[88] AppsSpider2. “InsightAppSec”;, [Online]. Available: https://www.rapid7.com/products/insightappsec/.

[89] AppsSpider3. “The Universal translator - 7 feature brief appsec universal translator”;, [Online]. Available: https://www.rapid7.com/globalassets/_pdfs/product-and-service-briefs/rapid7-feature-brief-appsec-universal-translator.pdf.

[90] Qualys. “Qualys was getting started guide”; Sep 2020, [Online]. Available: https://www.qualys.com/docs/qualys-was-getting-started-guide.pdf.

[91] Hassenklöver T.. “Teebytes TNT-Fuzzer”;, [Online]. Available: https://github.com/Teebytes/TnT-Fuzzer.

[92] KissPeter. “Kisspeter API Fuzzer”;, [Online]. Available: https:// github.com/ KissPeter/APIFuzzer.

[93] Gitlab. “Integrating security into your DevOps lifecycle”;, [Online]. Available: https://about.gitlab.com/solutions/dev-sec-ops/.

[94] SPIKE. “Fuzzer automation with spike”; Mar 2021, [Online]. Available: https:// resources.infosecinstitute.com/topic/fuzzer-automation-with-spike/.

[95] Vuagnoux M.. “Autodaf´e: An act of software torture”; 2005, [Online]. Available: http://autodafe.sourceforge.net/docs/autodafe.pdf.

[96] Godefroid P., Levin MY., Molnar D., Automated Whitebox Fuzz Testing. 2008;p. 151–168.

[97] WebGoat. “WebGoat/WebGoat: WebGoat is a deliberately insecure application”;, [Online]. Available: https://github.com/WebGoat/WebGoat.

[98] Digininja. “Digininja/ DVWA: Damn Vulnerable Web Application (DVWA)”;, [Online]. Available: https://github.com/digininja/DVWA.

BIOGRAPHIES

NAME	Miss. Chansida Makaranond
DATE OF BIRTH	14 July 2000
PLACE OF BIRTH	Bangkok, Thailand
INSTITUTIONS ATTENDED	Thai Christian School, 2017: High School Diploma Mahidol University, 2022: Bachelor of Science (ICT)

NAME	Mr. Pattarakrit Rattanukul
DATE OF BIRTH	4 Octorber 1999
PLACE OF BIRTH	Bangkok, Thailand
INSTITUTIONS ATTENDED	Suankularb Wittayalai School,2017: High School Diploma Mahidol University, 2022: Bachelor of Science (ICT)

NAME	Mr. Pumipat Watanakulcharus
DATE OF BIRTH	15 March 2000
PLACE OF BIRTH	Bangkok, Thailand
INSTITUTIONS ATTENDED	Triam Udom Suksa School, 2017: High School Diploma Mahidol University, 2022: Bachelor of Science (ICT)