Cyber Security Assessment Automated System

BY

NATTHA SIRIBOON

PAWEE TANTIVASDAKARN

SOPONPAKORN SUTTIKAO

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF ENGINEERING IN COMPUTER INNOVATION ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2019

SCHOOL OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

PROJECT CERTIFICATE

Project Title	Cyber Security Assessment Automated System
Student Name	Miss Nattha Siriboon Student ID. 60011240 Mr. Pawee Tantivasdakarn Student ID. 60011252 Mr. Soponpakorn Suttikao Student ID. 60011271
Degree	Bachelor of Engineering in Computer Innovation Engineering
Project Advisor	Signed: _____ (Asst. Prof. Dr. Orathai Sangpetch)
Project Co-Advisor	Signed: _____ (Asst. Prof. Dr. Akkarit Sangpetch)

Project Title	Cyber Security Assessment Automated System
Student Name	Miss Nattha Siriboon Mr. Pawee Tantivasdakarn Mr. Soponpakorn Suttikao
Degree	Bachelor of Engineering in Computer Engineering
Project Advisor	Asst. Prof. Dr. Orathai Sangpetch
Project Co-Advisor	Asst. Prof. Dr. Akkarit Sangpetch
Academic Years	2020

ABSTRACT

Security hygiene is very important, especially for the finance industry. For them, user privacy is one of the top priorities of their service. Security tests should be done regularly to ensure the highest security hygiene possible. Normally penetration tests are done by humans which is time-consuming and costly so it cannot be done regularly. This project aims to combat this problem. By using machines to run security assessment tests more often because the process can be automated. Our system can be run at any time of the day. We are using the black-box technique and open-source tools (Hydra, Dirb, Nmap, SQLmap, Amass, Searchsploit, and XSSsniper). Our client will give us permission and endpoints for us to run a penetration test. After our system runs all the necessary tests it will generate a report to send to our client. The report will include the information and vulnerabilities on the client systems.

ACKNOWLEDGEMENTS

Foremost, we would like to express our sincere gratitude to our advisors, Asst. Prof. Dr. Akkarit Sangpetch and Asst. Prof. Dr. Orathai Sangpetch, for their patience, constant support, and encouragement. Their guidance helps us a lot to learn from the problems that they point out and be able to continuously develop our capstone project.

Besides our advisor, we would like to thank the rest of my capstone project committee: Asst. Prof. Dr. Sutheera Puntheeranurak, Asst. Prof. Dr. Panarat Cherntanomwong, Asst. Prof. Dr. Chutimet Srinilta, Asst. Prof. Dr. Rathachai Chawuthai for their comments and questions.

Our sincere thanks also go to P' Mook, P' Sek, P' O, P' Moss, and P' Lhin for the discussion toward how to implement the system, their suggestions help us a lot on our decision.

Finally, thanks to each of us for the contribution and sleepless nights that we were working together.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS/ABBREVIATIONS

Symbols/Abbreviations	Terms
CIE	Computer Innovation Engineering
SIIE	School of International Interdisciplinary Engineering Programs
CSASS	Cyber Security Assessment Automated System

CHAPTER 1
INTRODUCTION

This chapter introduces the overarching themes of this report and places the motivation for the work into context. Thereafter, the rationale and goals defined for the investigation of the project are discussed, followed by a summary of the overall project. Finally, an overview of the dissertation is given on a per-chapter basis.

The internet has become a part of our daily life, a lot of services become easier ever before with just the phone and the internet, everything could happen by just you finger. The number of the users have growth, but the number of people with malicious intention also growth. Security could be a big concern when exploring the cyber world. Trading is also a service that has been change though time, it could now be used via the website or mobile phone application. With the amount of the money circulate with in the broker trading system, this field could be a great target for the hacker to exploit the system and steal the precious information from the system for various purpose e.g., steal information, inject a ransomware. Hernandez authored an article with an interesting information “62% of the apps sent sensitive data to log files, and 67% stored it unencrypted. Physical access to the device is required to extract this data.” [1]

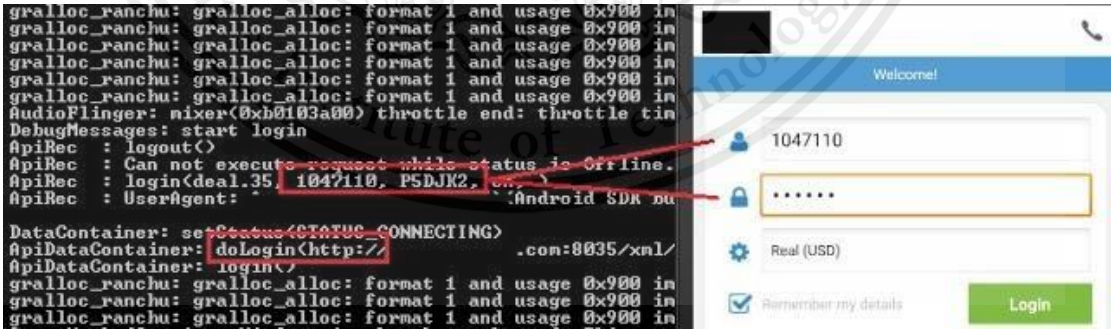

Figure1.1 Unencrypted log file, exposing the username and password.

1.1 Background

Proper configuration and best practice should be applied to the broker system to prevent the hacker to bleed into the system. One way to make sure is to hire the specialize company to perform a regular penetration test on the system, it could be

costly and time-consuming. With these cons, it was hard for many companies to hire and maintain their system with penetration test service. An unmaintained service could leak a various vulnerability.

1.2 Objective

Alternate to the previous solution, another approach could apply by using the black-box perspective. Scanning from the internet is not rich in complexity of the scan test options compared to the on-site penetration but it leaves you with the convenient, to be able to scan whenever you want.

CSASS aim to be the automate service which will continually schedule scan the give broker endpoints. With the objective of

1. Continuous access the scanning services.
2. Automate the process of the scan and report.
3. Be able to scale.

1.3 Scope

- Auto generate report for the company.
- Deploy it in the sense of be able to scale.
- Perform an automated scan with the given endpoints from the company with the black-box perspective.

1.4 Method

- Research and analyze the strength of each tool and learn the process of penetration tests.
- Design the System Architecture and Database
- Design the system by integrating those tools.
- Test the designed model and collect information.
- Deploy the designed model and penetrate the customer's security system.
- Identify the vulnerabilities.
- Collect all the log files and send a report to the company.

- Run a sprint by going through the 2. to 7. Again
- Visualize to the consumers.

1.5 Expected Outcome

- Client companies gain some knowledge about their system's hygiene. though report and continuously improve their system.
- The scan will be able to easily run and perform automatically.

1.6 Table of Operation

Table 1.1 Operating Time

Topic	Month									
	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun
1. Preparation										
1.1 Research and find suitable tools for inspecting vulnerabilities.										
1.2 Find examples of systems and report for reference.										
1.3 Design architecture and database										
1.4 Design the first attack on the first vulnerable.										
1.5 Create Kubernetes one node cluster.										
1.6 Create cloud Postgresql cluster.										

Topic	Month									
	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun
2. Implement Application										
2.1 Upload each tool image and report generator										
2.2 Implement Argo workflow report for reference.										
2.3 Implement Argo workflow template.										
2.4 Implement data cleaner.										
2.5 Deploy web server for user interaction										
2.6 Implement backend service.										
2.7 Implement Argo middleware service.										
3. Testing and debugging										
3.1 Test and improve the performance of the workflow.										
3.2 Test and debug each tool.										
3.3 Test and debug workflow										
3.4 Test with provided endpoints by the company.										
3.5 Get feedback from the company										
3.6 Debug workflow after applying web										

application security risk tools.										
3.7 Test with provided endpoints by company II.										
3.8 Test, Debug, and improve the performance of the workflow.										

1.7 Report Outline

The rest of this report is organized as follows:

Chapter 2 reviews the state-of-art in vulnerability assessment
Chapter 3 describes the design and implementation of automate security scan system.

Chapter 4 demonstrates how various tools have been validate and the result of system evaluation.

Chapter 5 closes the report, reviewing the work undertaken and draws conclusions about key parts of the work that was undertaken.

CHAPTER 2

Literature Review

This chapter discusses the state-of-the-art of security vulnerabilities and tools considered during the analysis and design phase of this project. The investigation served 3 purposes: firstly, we wished to research Theoretical Background (section 2.1); secondly, we wished to identify the tools that will be use (section 2.2); Thirdly, we wished to research on previous related work on this field (section 2.3); Finally, on (section 2.4) summarize the chapter.

2.1 Theoretical Background

- **Detection of XSS**

As shown by a study [17], Cross-site scripting has been used for several types of attacks, such as Phishing attacks, Steal Cookie, DOS attacks, DDOS attacks, and XSS worms. It's possible to inject on whether the client DOM or the server using different techniques. To detect the exploit, a different method could be used. Static Analysis Methods apply by analyzing codes of the application. While Dynamic analysis methods detect by injecting data into the website and observe whether an attack is triggered. In our project, with black box scope where can't access the client source code, Dynamic analysis methods are more appropriate.

- **Index Calculation**

The index is calculated from found and impactful factors and is classified by level to measure cybersecurity and then produce the total index more accurately. This index would express and conclude the overall system which tells the condition, how current status on going, and for instance, if, one of the indexes, shows below standard then the system would tell how and where to prevent any exploits. [15]

- **Security Vulnerability Category and Blackbox testing**

There is various type of the security vulnerability out there, some of them can attack only if the person could access the devices while some of them could attack from the internet. According to the OWAP top 10 web security risk [16], Not all of them could efficiently test via the black-box perspective e.g., Broken Access Control, Security Misconfiguration, Insecure Deserialization, and Insufficient Logging & Monitoring. OWAP also provides a vulnerability image for testing called OWAP juice

shop [3] which could be installed on any machine web server and testing with the different difficulty of security setup.

- **Common security vulnerabilities**

The two most common security vulnerabilities are SQL injection and cross-site scripting. 42% of the attacking goal is to gain sensitive information. Most of the vulnerabilities are easy to be exploited and do not require advanced techniques or skilled attackers to exploit them.

- **SQL injections**

The most basic and common way is to put '1' OR '1' = '1' into username or password, then press enter. If the target application is vulnerable to this attack, the attacker will gain access to information in the target database. [18] If the target application is immune to basic SQL injections. Blind SQL injections can be used to gain more information about the database. Blind SQL injection is a way to ask database true/false questions and gain information that way. [19]

- **Network topology**

The network reconnaissance usually checks the target IP's status, which, currently active or not and Nmap is one of them. Normally, Nmap would ping ICMP protocol to the target IP by sending echo request to port 80 and if they received a signal that would let the Nmap know that which ports are still active or go down and even detailed information from the target IP.

2.2 Tool Stacks

In our project we classified tools and implementations, which were used, into 6 methods: Cloud-structure, Front-end, Back-end, Workflow, Generates a report, and Deployment and Testing.

2.2.1 Tool for cloud-structure

- **Digital Ocean**
  One of the largest cloud platforms with the best price for CPU performance compared to the other competitors. For this project, we use 7 of their services which are Kubernetes, Droplet, Networking, Container Registry, Spaces, Databases
- **Kubernetes(K8s)**

A system for automating deployment, scaling, and management of containerized applications. For this project, Kubernetes was used for most of our services that are containerized.

- **Droplet**

A cost-efficient virtual machine with the ability to run the existing image or private image. For this project, Droplet was used by Kubernetes to create a Node Cluster

- **Networking**

Networking includes many services such as Domain, FloatingIPs, Load Balancers, VPC, Firewall, PTR record. For this project, FloatingIPs was used to reserve a static IP, VPC for the private network, and Load Balancers was used by Kubernetes resources, combined as NGINX Ingress Controller using a reverse proxy to the services.

- **Container Registry**

A private repository that lets you host your private images with the version record on the cloud. For the project, Container Registry was used to host the workflow containers images.

- **Spaces**

Object storage with S3-compatible and built-in CDN. For this project, Spaces was used for storing the log files output from each tool in the workflow, and the report.

- **Databases**

A managed database cluster with the options of PostgreSQL, MySQL, and Redis. For this project, the Postgresql was used for a storage system.

2.2.2 Methods for Front-End

- **NuxtJS**

A powerful framework for simple and powerful web development. Building on top of Vue.js and supporting SSR. For this project, NuxtJS was used for making a web application

- **Keycloak**

Keycloak is an open-source Identity and Access Management solution aimed at modern applications and services. It makes it easy to secure applications and services with little to no code.

2.2.3 Methods for Workflow

In this Project, the tools from Kali Linux [13] have been used.

- **Argo Workflow**

Argo workflow is a workflow engine that runs as a container native. It could easily be integrated with Kubernetes. Argo allows you to orchestrate highly parallel jobs, in which each step is built from a container as a multi-step or DAG in the workflow without the overhead of legacy VM. For this project, Argo was installed as a Custom

Resources Definition in Kubernetes and used to run the workflow (from the template) as a DAG with various tools.

- **Nmap**

Scan at the communication level of protocols to find the status of ports that get scanned such as TCP and UDP port, scan in secret to make only a two-way handshake (normally scan is a three-way handshake), and scan OS information of the target IP or domain name. Able to use scripts within Nmap's resources such as vulnerability script.

- **SQLmap**

Used for detecting and exploiting SQL injection flaws. SQL injection flaws can be exploited by inserting SQL statements as user input to manipulate databases. If the target does not protect itself against SQL injection, SQLmap can gain data in said database.

- **Amass**

Scan for IP addresses and all subdomains were given a domain name.

- **Searchsploit**

Search known vulnerabilities in OS or platform. Searchsploit stores a copy of the exploit database in local storage for quick search. It needs to be updated once every two weeks to get the most up to date information.

- **CVE**

A list of records that contain an identification number, a summary, and at least one public reference—for publicly known cybersecurity vulnerabilities. CVE Records are used in numerous cybersecurity products and services from around the world, including NVD.

- **Hydra**

Brute force user and password discovery from various protocols such as ssh, HTTP, POST.

- **Dirb**

A web scanner, scan for available directories from the commonly used word lists.

- **XSSSniper**

Crawl for a form and scan if it is available for cross-site scripting.

2.2.4 Method for back-end

- **Postgresql**

The database will be connected to the API server and any service needs to access the database via API. Some tables in the database will need admin privilege to insert, update, or delete. This is to ensure maximum security. We also have a backup database in case the main database is down.

- **Go-gin**

Golang framework used to make an API server. This framework is very fast (40 times faster than martini). It manages to get packages for the programmer. It has a built-in net/HTTP library for Golang. The library contains almost every function needed to do routing and rendering. The framework has good error management. [14]

2.3 Related Work

- **Scoring System for Vulnerability**

The Common Vulnerability Scoring System (CVSS) [12] expresses a way to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity. The numerical score representation in the enumeration (such as low, medium, high, and critical) to help organizations properly assess and prioritize their vulnerability management processes.

- **Penetration Test and Security Assessment Report**

One of the first factors is that the summary of vulnerabilities encountered in the system shows how many severities for a company to see and investigate details after scanned. [7]

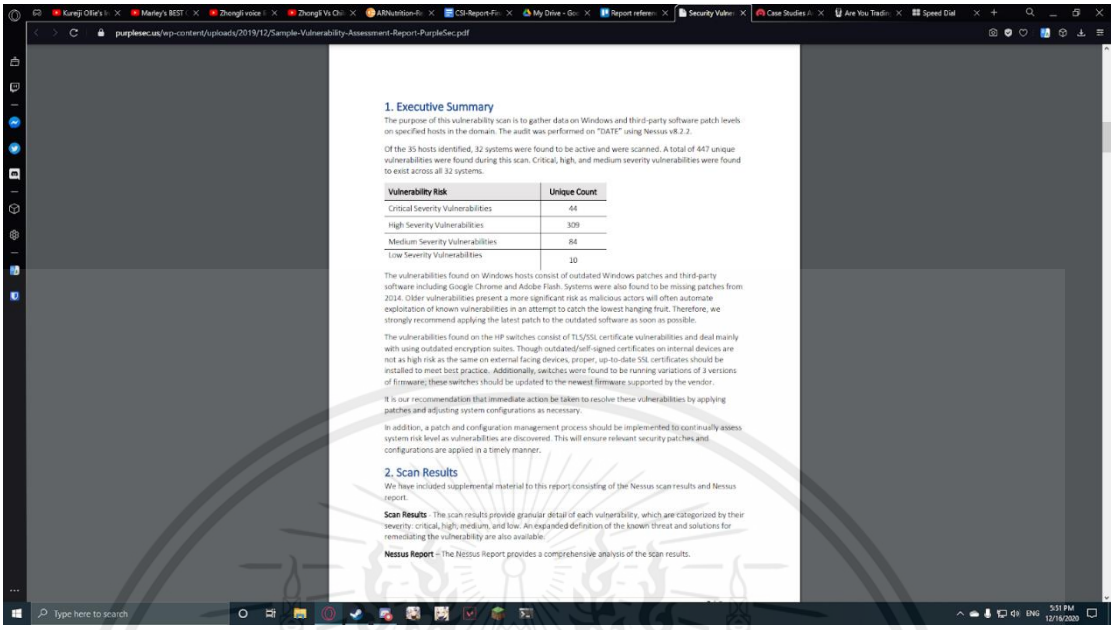

Figure 2.1 The counter of severity in the target system.

- **Penetration Test and Security Assessment Scanner**
  A scanner on web applications shows an example of a scanner that can produce a report and analyze the result. [11]

2.4 Chapter Summary

In Chapter 1 we proposed the important of the cybersecurity related to the finance, especially, broker field. We also proposed how our CSAAS could improve this eco system.

In this chapter the state-of-the-art was categorized into Theoretical Background (section 2.1), Tool Stacks (section 2.2), Related Work (section 2.3). Observations were made on the various theories (section 2.1), tools and its definition has been defined (section 2.2), and the relevance of the previous works was review (section 2.3).

The next chapter presents the design of CSAAS, which is a system intended to automate the cyber security assessment.

CHAPTER 3
METHODOLOGY

3.1 Introduction

In Chapter 2 we identified that automated the security scan with the black-box perspective could be an interesting solution compared to the penetration test.

This chapter describes the design of CSAAS, a system that assess the vulnerabilities automatically. First, the chapter describes the vulnerabilities assessment approach (section 3.2), and system requirements (section 3.3). A Design and diagrams (section 3.4-3.5). The storage technology (section 3.6). A proposed flow is then discussed in section 3.7.

3.2 Vulnerability Assessment Approach

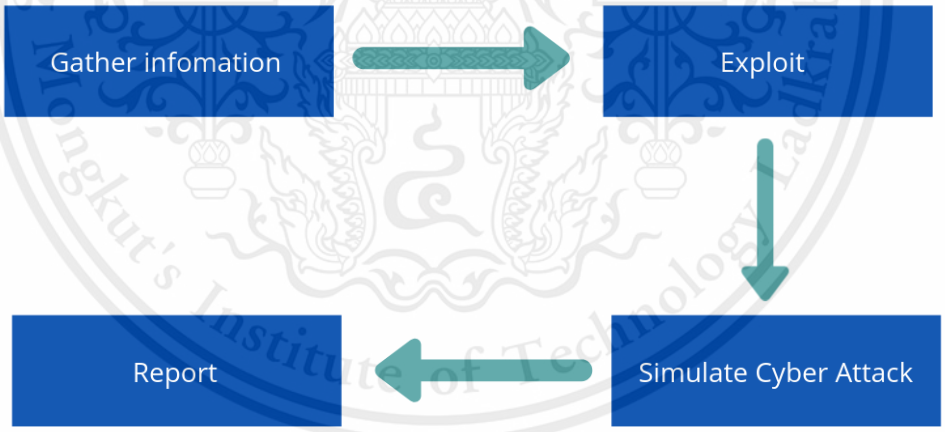

Figure 3.1 Steps of Vulnerability Assessment

The step should be start with gather the overall general information before digging deep down to the exploit and start the assessment and report as in Figure 3.1. From the article [d], After we researched and familiar with each tool then we can design our first flow in three steps: Reconnaissance or investigating the target and retrieve information from it by using Nmap, for protocols, and Amass, for the subdomain. Exploitation or search vulnerabilities in the system including find directory, brute force

list of usernames and passwords, and. The report, we have the reportable to be generated by Carbone that converts into a PDF file as in Figure 3.2 And in the future, we will add other features such as index calculator.

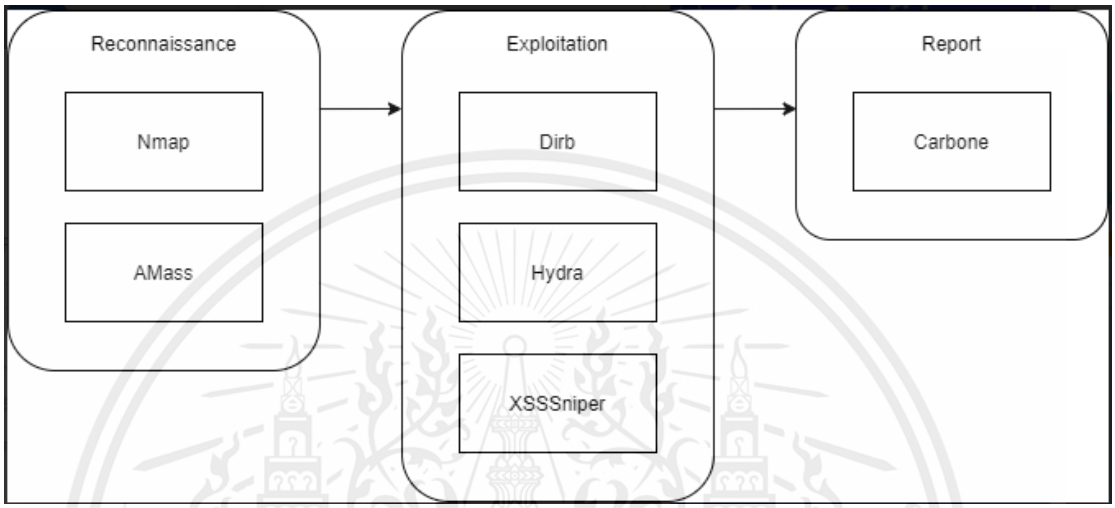

Figure 3.2 Scan Method used in the project.

3.3 System Requirements

- Able to accurately identify target's vulnerabilities.
- Be able to test in each specific window of time.
- Able to perform test automatically.
- Able to produce security assessment report in the form of pdf file.
- All services are secured.
- Easy to maintain.
- System is scalable.
- System is run on cloud.
- The microservices are stateless.
- Users can login to the system using username and password.
- Users can insert and delete endpoints form their endpoint list.

3.4 Overall Architecture System

Figure 3-2 Show the overall architecture of the CSI system the user accesses the website csi.cmkl.ac.th on their browser. The domain name is resolved by Cloudflare

DNS service and points to our digital ocean Kubernetes resources. Every request from the user's browser would go to the Ingress NGINX Controller, the integration of the Nginx on Kubernetes with digital ocean's load balancer and pass the request to the service which either Web Server or Backend services depend on the path of the request. During the signing in, the web browser would redirect to Keycloak's OAuth. Keycloak's authentication would then be used on the backend services to authorize the user on the Keycloak server.

Users are allowed to add, modify, and delete their endpoints that they want to scan on the browser. The record will be stored on the PostgreSQL cluster via the backend service while the scan booking schedule will be created by admin and the schedule as a cron workflow and submit to the Aro Server via API from the Backend service. When the time comes, Argo would execute the scanning template according to the cron workflow and save the log of each tool to the digital ocean space object storage and report the scanning status to the database.

The finished workflow will create the ready-to-generate JSON on the database. The user now could fetch their report on the website which will pull your JSON data and create the report directly from the Report generator.

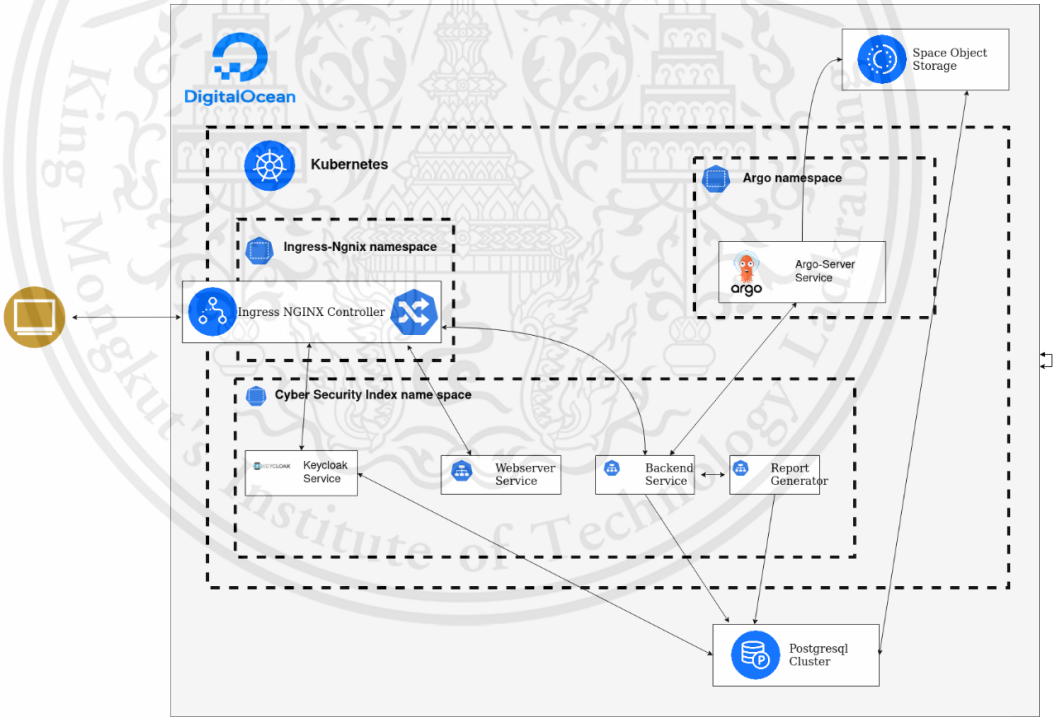

Figure 3.3 Overall System Architecture

3.5 Diagram

3.5.1 Use Case Diagram

As shown in Figures 3-3, before the user starts using the system, first the user will have to get the whitelist. The user starts with Sign in with Google Account with Google OAuth Authentication. This method will verify the identity of the user. Eventually, after being verified, the user will be able to Add Scan Endpoint, Edit Scan Endpoint, Delete Scan Endpoint, Book Scan Schedule, Get Scan History, and Download Report.

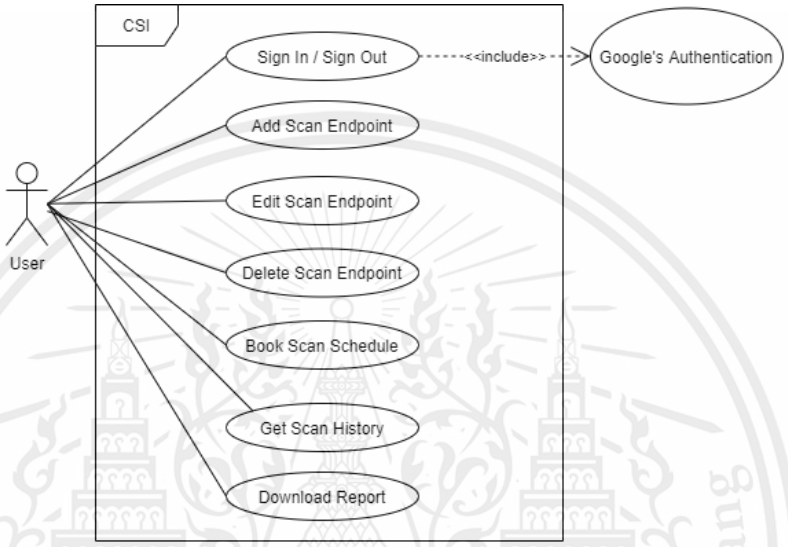

Figure 3.4 Use Case Diagram of the system

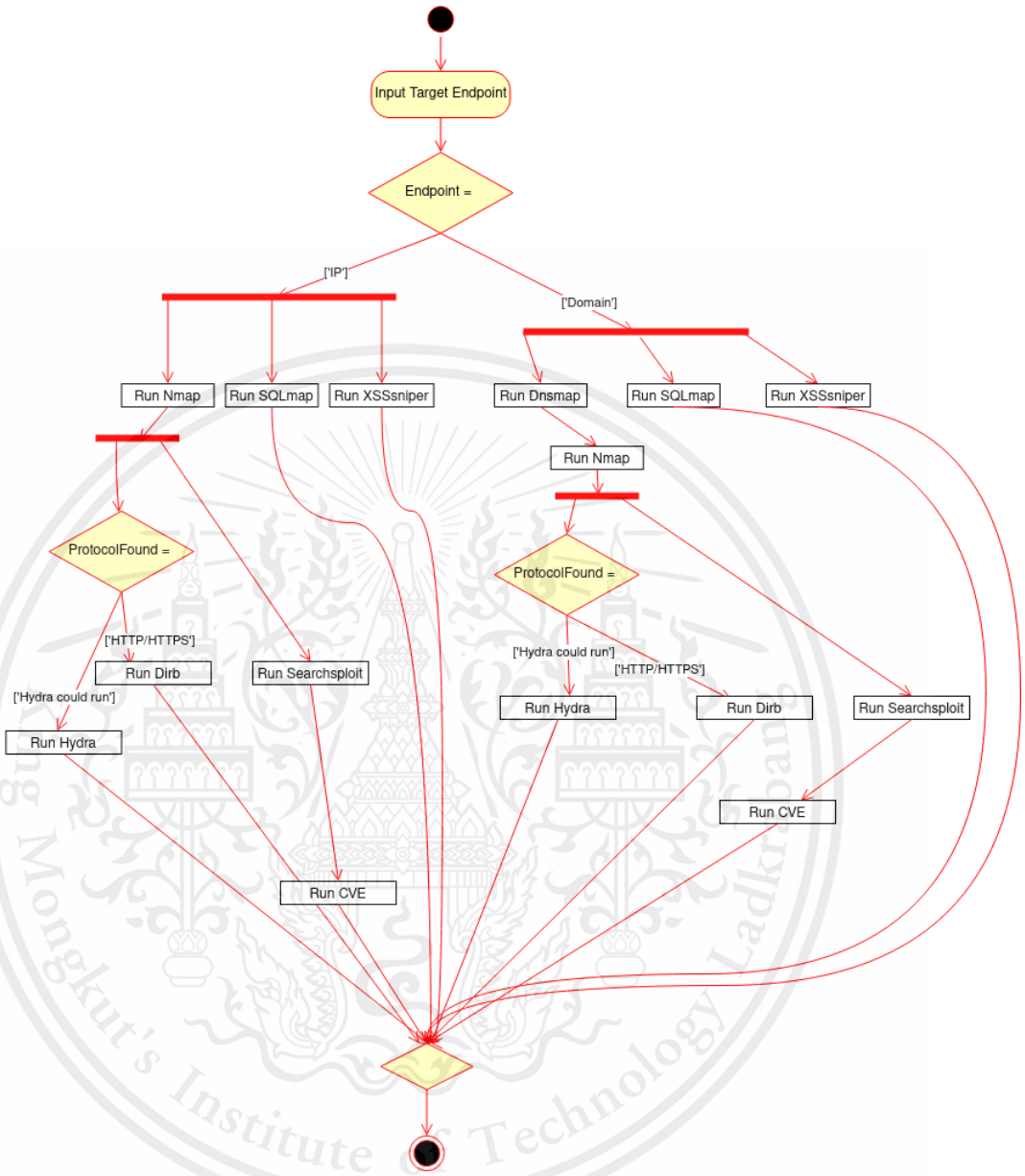

Figure 3.5 Activity Diagram of the scanning workflow with a single endpoint

3.5.2 Activity Diagram

The flow of the Argo workflow is illustrated in Figure 3-4. After the receive the input 'Endpoint' it will check whether the endpoint is a domain name or IP. The difference is if the endpoint is a domain, it will run Amass. otherwise, it will run Nmap. In both conditions, the tool will run along with other tools which are SQLmap and XSSsniper in parallel. The result of the Amass will be processed by the Nmap. The tools that run after the Nmap are Searchsploit that

will use the OS info to find the CVE entries which will then be processed by the tool CVE. Searchsploit runs in parallel with hydra or enum4linux depending on the result of the Nmap. the result of enum4linux could also be processed by the Searchsploit.

3.6 Storage

3.6.1 Object Storage

The full log file should be kept for the further audit, the structure designs follow Figure 3. This ensures that the owner of the hierarchies for the companies, each scan, each endpoint, and each tool.

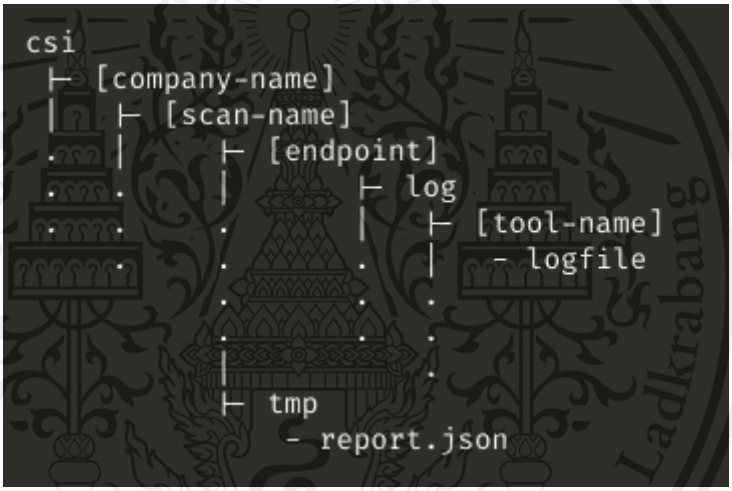

Figure 3.6 Object Storage Structure

3.6.2 Database

The database is used to store results from scanning and user accounts information. Because results from each tool come in different formats, they will be stored in the Postgres No-SQL database. Each record will contain the name of the tool it came from, scan ID, endpoint point it scanned, and result. This information will be used to create a report for our customer. Use accounts information will be stored in the Postgresql database. SQL databases consist of four tables. All databases run on the digital ocean.

Table 3.1 List of scans

Attribute	Data type	Description
scan id	varchar	unique id for each scan, surrogate key. PK
company	varchar	Owner of scan
status	enum	The status of scan has four possible values: failed, success, running, terminated, suspended. failed: scan has failed. success: scan is done successfully. running: scan in-process, scanning. terminated: scan is terminated by the user. suspended: scan in-process, a scan has stopped waiting to resume.
Start	Time stamp	Start scanning time
complete	Time stamp	Completed scan time

Table 3.2 List of endpoints

Attribute	Data type	Description
endpoint	varchar	endpoints that users own.
company	varchar	owner of endpoints.

Table 3.3 List of reports

Attribute	Data type	Description
Scan id	varchar	Produce from which scan
DIR	varchar	Directory that keep the report

Table 3.4 List of results

Attribute	Data type	Description
Scan id	varchar	Result of which scan
Data	JSON	Data from tools

API section

Table 3.5 List of APIs

APIs	Description	input	output
/addEndpoint	Allow POST method, used by user for adding a new endpoint.	Header contains user's token JSON body contains endpoint	-

/addScan	Allow POST method, used by our system for creating a new scan.	JSON body contains: ScanID (auto generate by our system), company's name of its owner	-
/updateScan/:status	Allow PATCH method, used by our system for updating scan status	The parameter contains: status. JSON body contains: scanID and status	-
/getAllScans	Allow GET method, used by user to get all their scan information	Header contains: user's token	List of scan along with corresponding information: scan id, status, start time and end time.
/getEndpoints	Allow GET method, used by user to get endpoints in a user's endpoint list.	Header contains user's token The parameter contains: google id of the owner of the endpoint list.	list of endpoints
/DeleteEndpoint	Allow DELETE method, used for deleting endpoint from user's endpoint list.	Header contains user's token JSON body contains endpoint	-
/uploadResult	Allow POST method, used by our system for upload result of scan in JSON format	JSON body contains scan id and result of scan in JSON format	-
/getAllEndpointsAdmin	Allow GET method, used by admin to get all registered endpoints in system	-	List of all endpoints in the database and its owners.
/generateReport	Allow POST method, used by our system to create report from scan	JSON body contains scan id	Security assesment report in pdf file.

3.7 Summary

This chapter described the high-level requirements and design of a system that CSAAS. The chapter started by describing the approach of vulnerability assessment and discussed more about the design of the system.

The System features is covered in further detail in Chapter 4 which describes the results of the implementation and experimental.

CHAPTER 4
EXPERIMENTAL RESULT

4.1 Introduction

Chapters 3 and 4 described the design of CSAAS, a system that assess the vulnerabilities automatically. In this chapter, we present an implementation and testing method and its results that show the functionality of the system, a tools validation, and evaluate the overall system performance against the test. The chapter is organized as follows: (section 4.2) walkthrough the functionality of the implemented system; (section 4.3) measure the reliability of tools; (section 4.4) evaluate the system against the real-world environment.

4.2 Functionality of the System

4.2.1 Backend API

We use tokens to identify users. Users will be able to gain access only to their endpoints list and their scans history perform some function (add, remove, update etc.) on their endpoint list. Some functions will be performed by our system. We use Go-gin and GORM to connect with Postgresql database. The figures below will show functionality testing of our backend API.

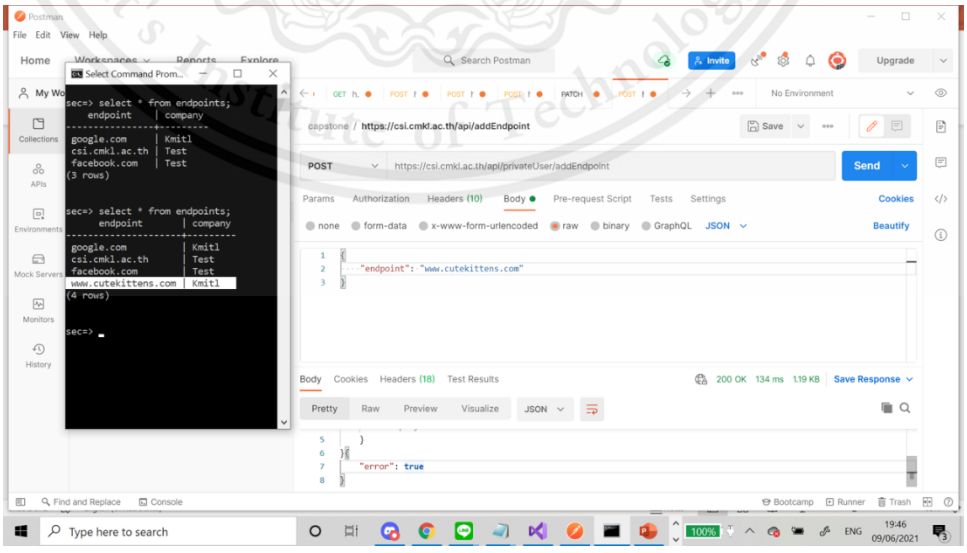

Figure 4.1 Add new endpoint to user's endpoint list. The command line show database before and after insertion, new endpoint highlighted in white.

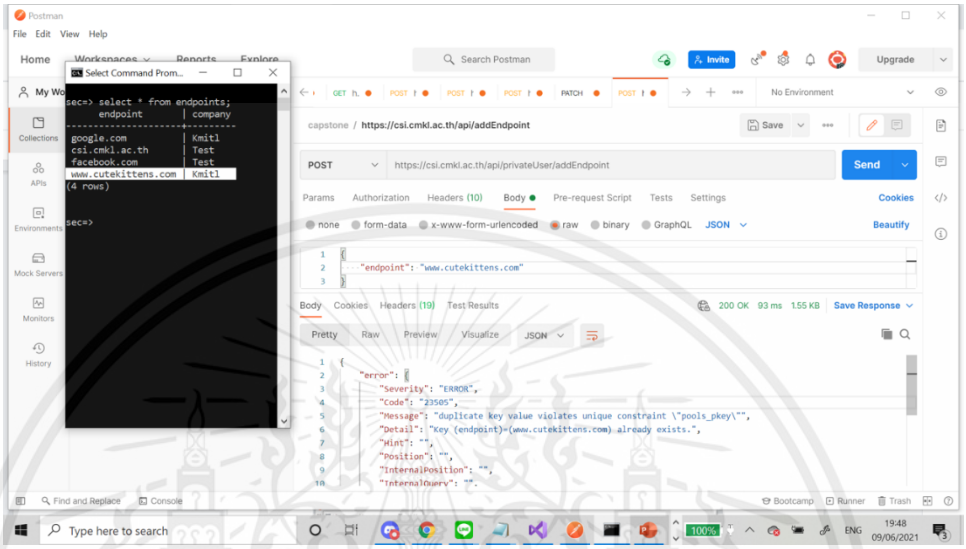

Figure 4.2 Return error (primary key conflict) after an attempt to insert existing endpoint.

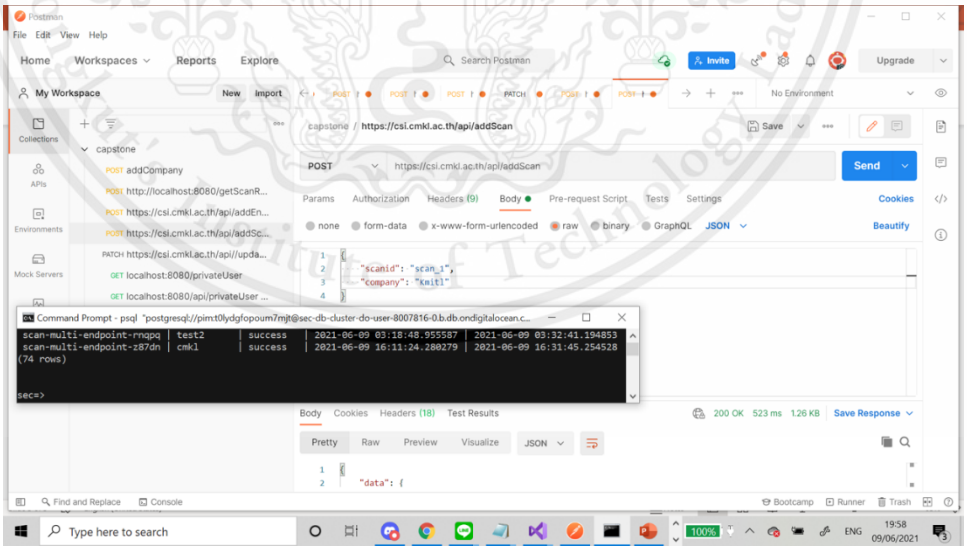

Figure 4.3 command line show database before adding new scan. This process will be done by our system.

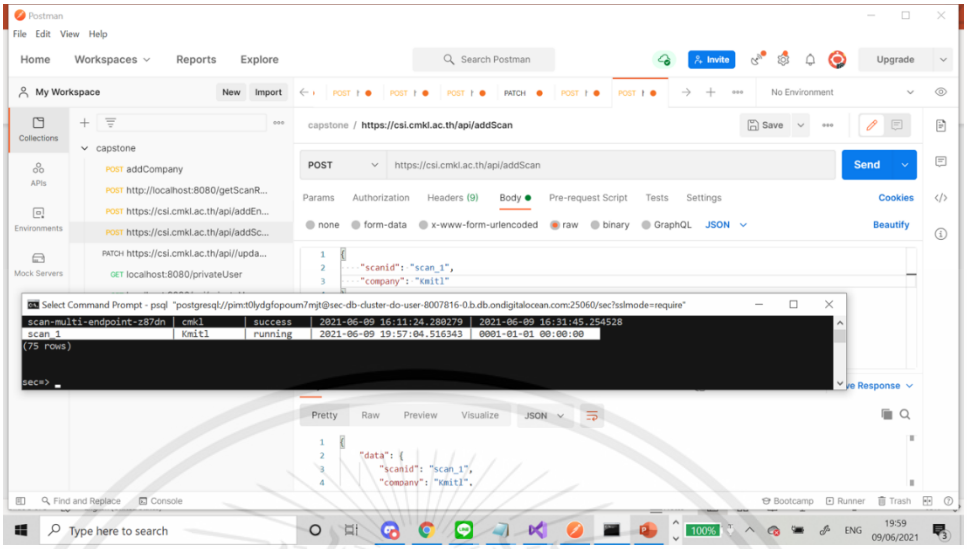

Figure 4.4 Database after adding a new scan. Scan id will normally be autogenerated, but for the purposed of demonstration, we will add new scan using scan id ‘scan_1’

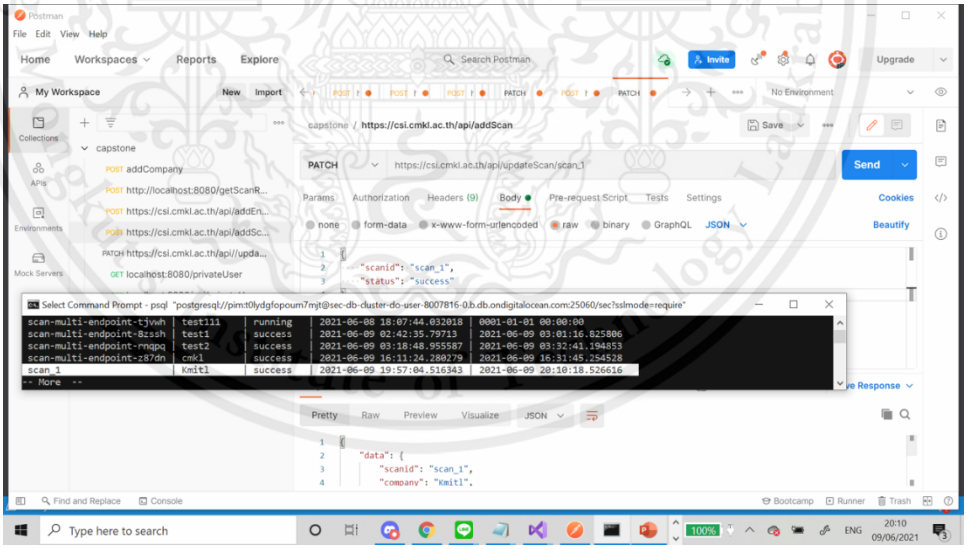

Figure 4.5 After changing status of ‘scan_1’ from running to success the competed time stamp change from default value to current timestamp.

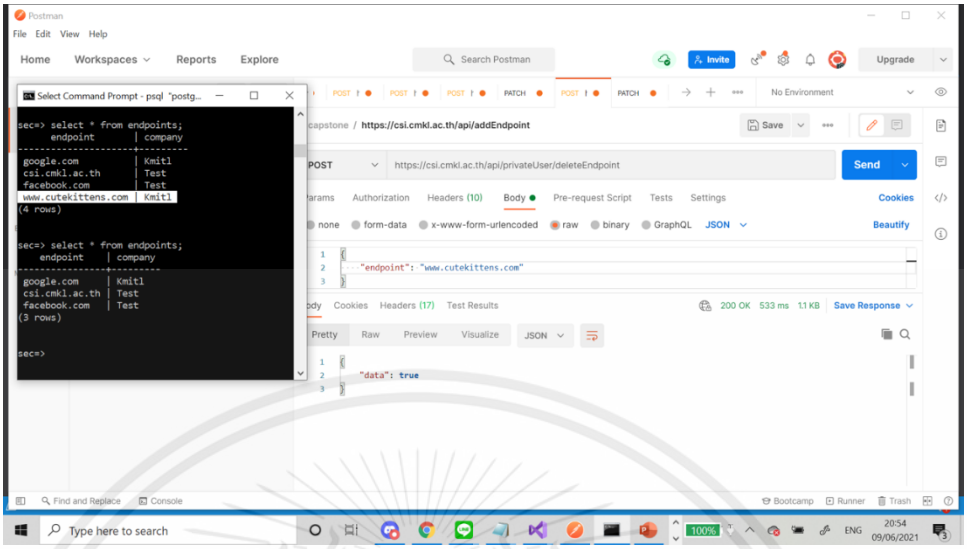

Figure 4.6 Deleted endpoint highlighted in white.

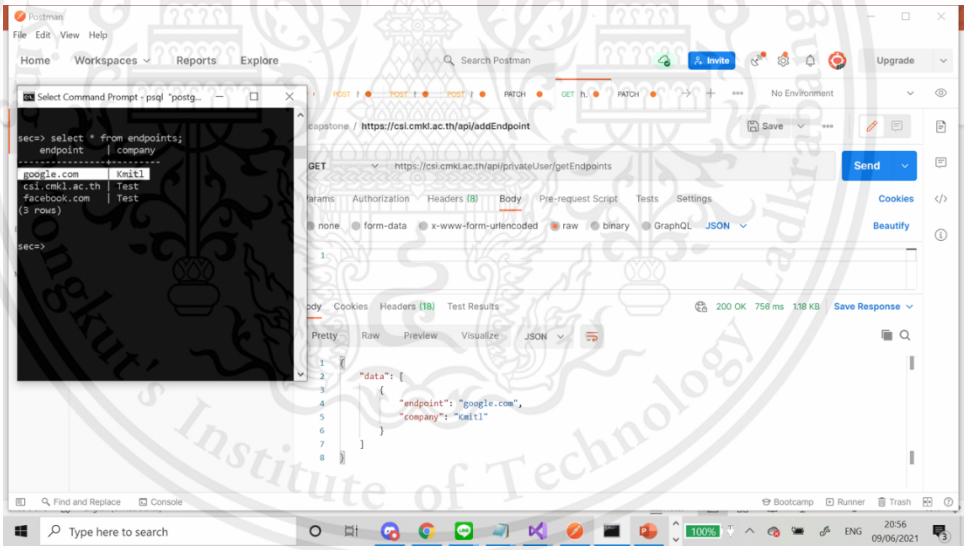

Figure 4.7 User get their endpoint list.

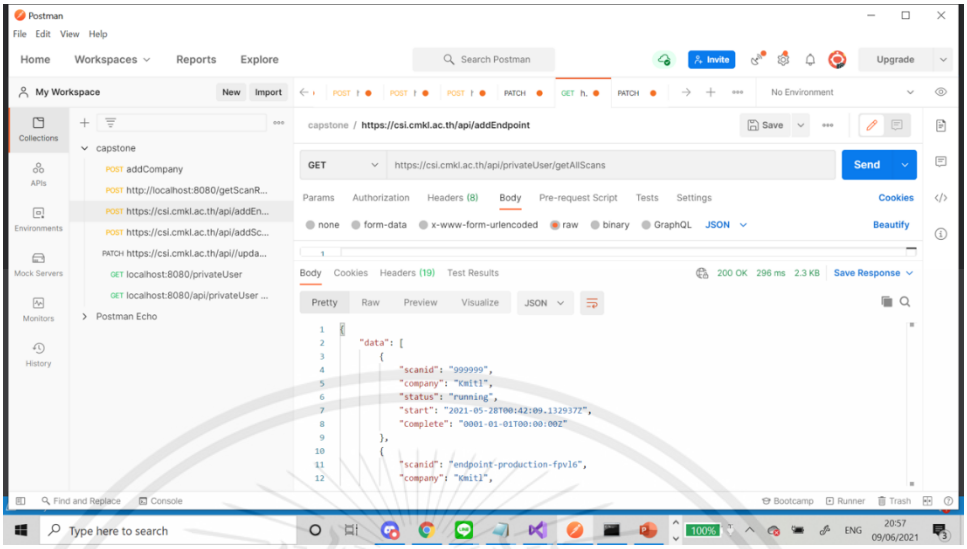

Figure 4.8 User get their history of scans.

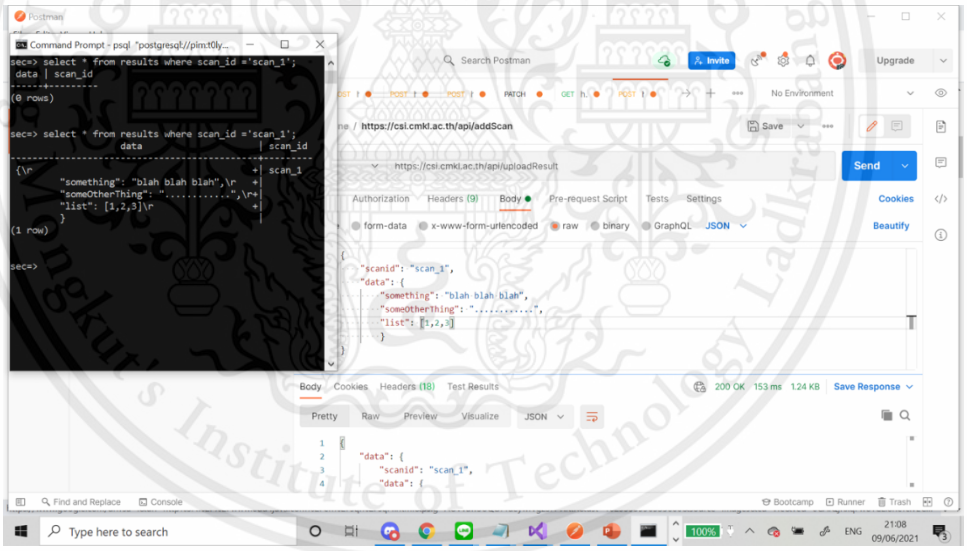

Figure 4.9 upload scan result to database. The upload was successful, and the command line showed database before and after upload.

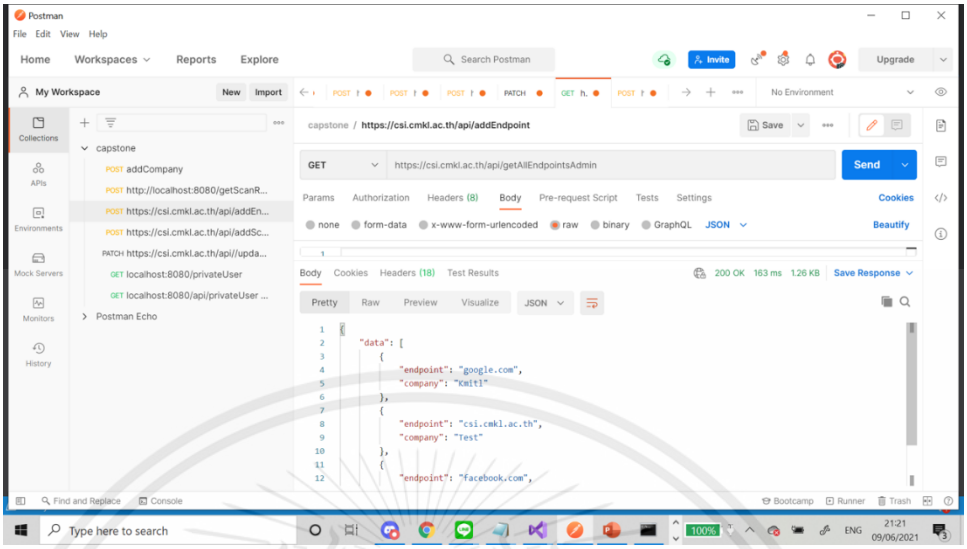

Figure 4.10 admin gain access to information of all user endpoint lists.

Figure 4.11 generate report from scan, return pdf file.

4.2.2 Front End

User could log in from the frontend page. This will be redirected to /auth path which will handle the Keycloak's OAuth Authentication (Figure 4.12). After login, the user will receive a token which identify the use role and group (assign from the Keycloak admin page) Using this token, the user will land on the page for their role as in Figure 4.13 for company and Figure 4.15 for admin. The user in the company could add some endpoints and these will be shared in the pool within the company (Figure

4.13) and access the scan history as in Figure 4.14 While the admin could see all request endpoints from all companies (Figure 4.15)

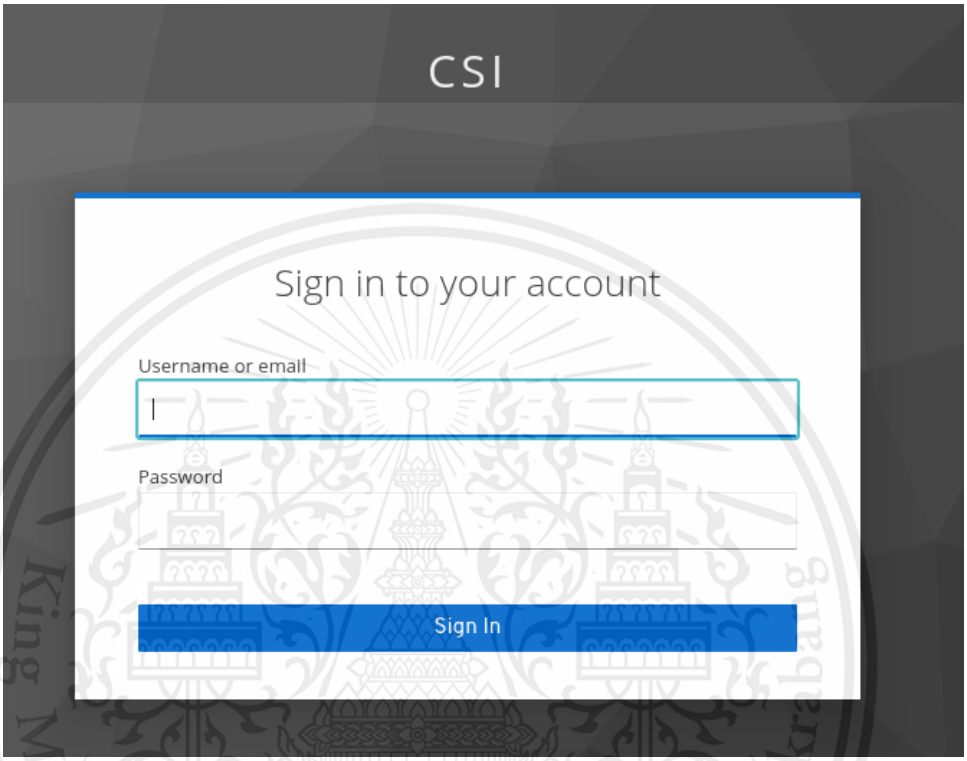

Figure 4.12 OAuth login page

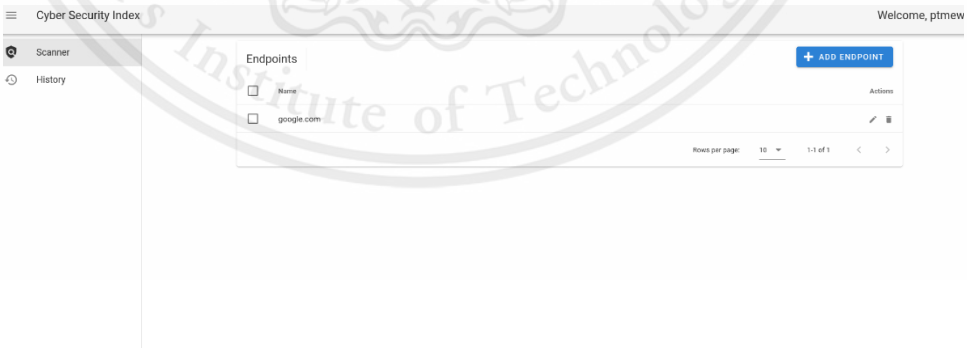

Figure 4.13 Company scanner page

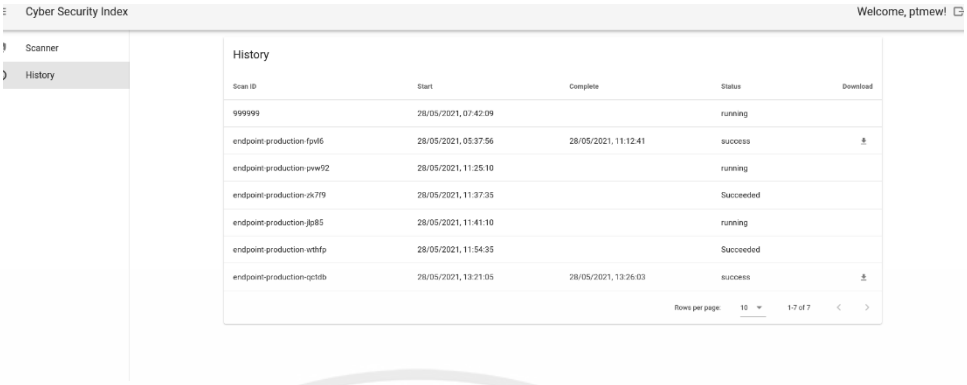

Figure 4.14 Company history page

Figure 4.15 Admin page

4.2.3 Workflow

Argo Workflow could be written in the form of the template for the workflow that is repeatedly invoked. We create tools templates along with scan one endpoint template and scan multiple endpoints as in Figure 4.16. Digital Ocean Container Registry used to store the container image which templates could pull as in Figure 4.21 The details of scan one endpoint and multiple endpoints in Figure 4.18 and Figure 4-19 sequentially. The result logs would save in the Digital Ocean Space as in Figure 4-20.

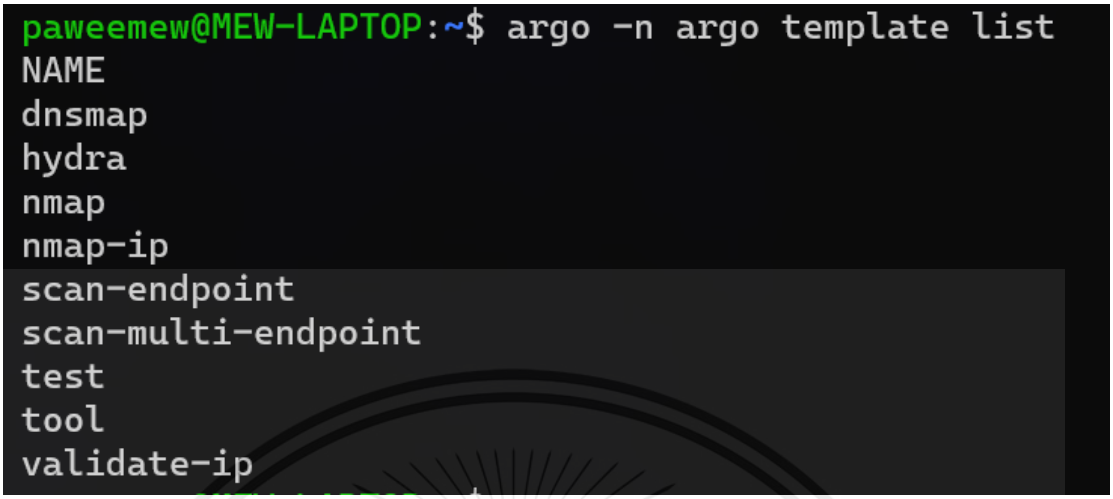

Figure 4.16 Ago Template YAML file was submitted to the Kubernetes and stored in the Argo namespace. It could be viewed from the Argo CLI.

Figure 4.17 Ago Template YAML file was submitted to the Kubernetes and stored in the Argo namespace. The template could be viewed as a GUI, alternatively of Argo CLI on Figure 4-10, using the port-forwarder command of kubectl to port forward your service from the Kubernetes to your localhost.

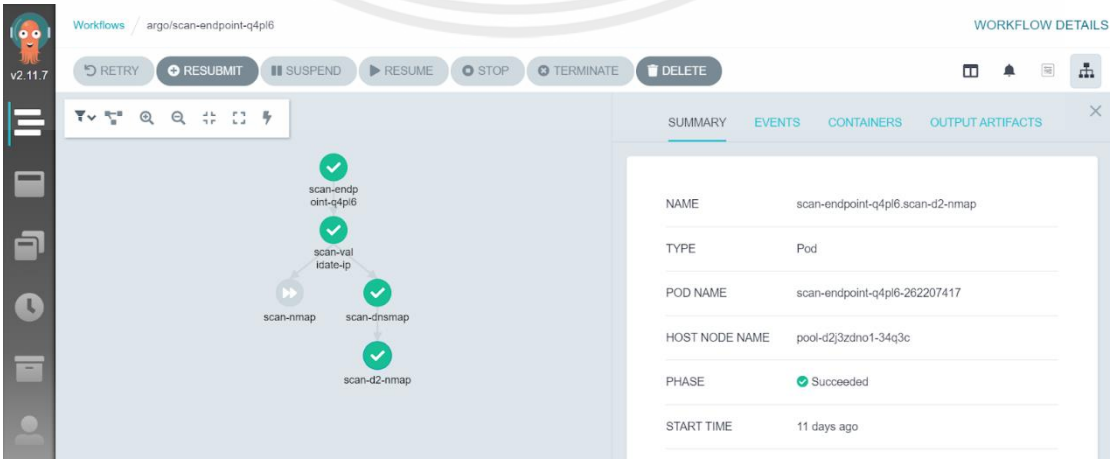

Figure 4.18 Argo submitted scan one endpoint from the template on the Argo server client. The scan process could be in the form of submitting the template, the scan one endpoint allows you to submit the form with the parameter of the target endpoint. On this figure, the input was a domain name, so the flow ran the DNS map then Nmap rather than just Nmap.

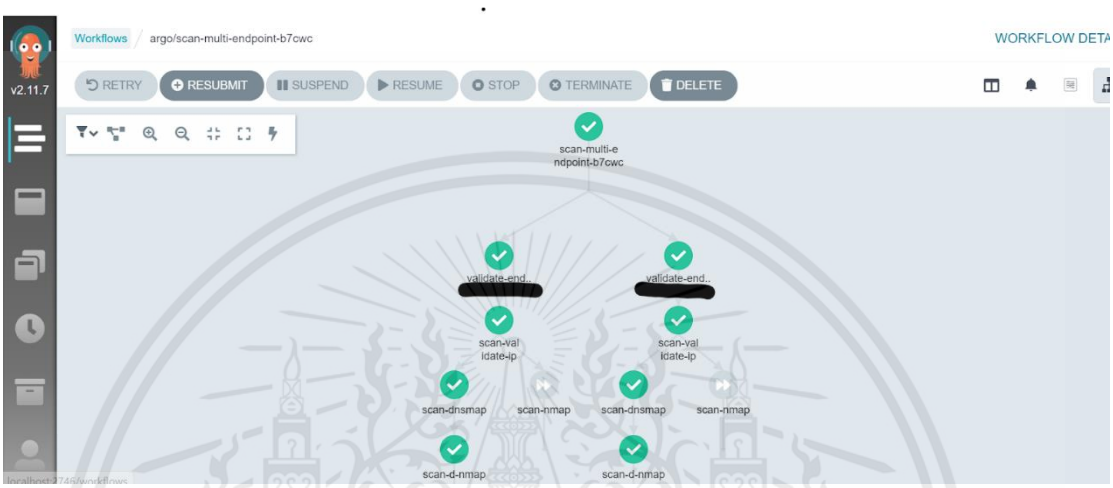

Figure 4.19 Argo submitted scan multiple endpoints from the template on the Argo server client. The template was built on top of the scan one endpoint template so it could get the list of the endpoints and invoke one endpoint template in parallel.

Figure 4.20 Digital Ocean Space where the log file from the scan process would be saved.

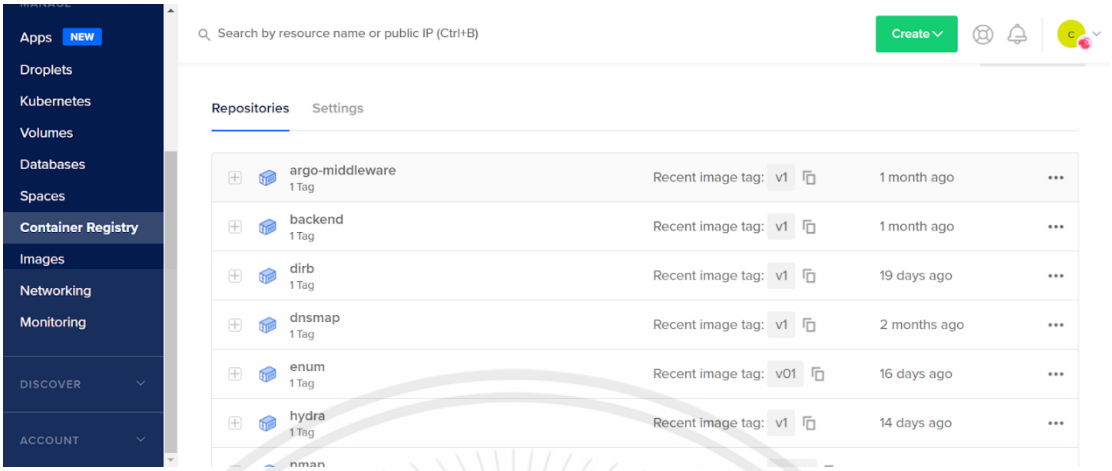

Figure 4.21 Digital Ocean Container Registry were used for storing the private container images for the Kubernetes Deployment and Argo Workflow.

4.2.4 Filtering and Report Generator

After all scan finished, the filter service will mount and recursively search for the log file and filter it for each service to a pack of JSON (Figure 4.22), ready to use for the report generator, Carbone service.

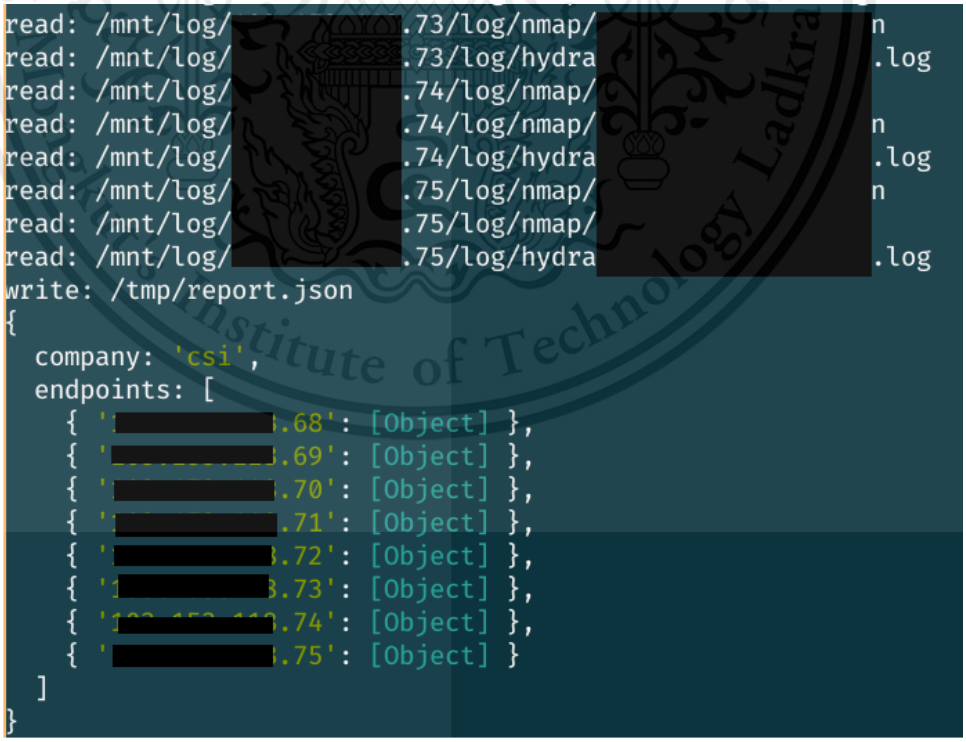

Figure 4.22 Output from the terminal showing the process of filtering the data from the log file.

4.3 Tool Validation

We first determine if the tools we choose to work with produce accurate results. We do this by running the tools against target with known vulnerabilities to see if the tools correctly identify target's vulnerabilities. If the tools can successfully identify said vulnerabilities it will mean the tools are accurate. We also Dubble-check by manually check the result the tools produce. We also measure the system resource usage (section 4.3.6)

4.3.1 Dirb

The tool has been validated by using the differences 2 set of testing on the testing Apache server. First, create files in directory with different file permissions and settings such as redirect the page as on the Figure 4.23 Second, create files in directory that return the different response as on the Figure 4.24.

File	Folder	Permission	Response
Root		644	200 OK
admin		0	403 Forbidden
adamin.php		644	500 Internal Server Error
auth		666	200 OK
index.html		666	
	com2	0	403 Forbidden
	com1	644	
com1/1		644	200 OK
com1/2		644	200 OK
com1/3		644	200 OK
com3			302 Temporary Redirect

Figure 4.23 Testing on files with different permissions.

Reponse	File	Result
100 Continue	100.php	100
101 Switching Protocols	101.php	101
103 Early Hints	103.php	500
200 OK	200.php	200
201 Created	201.php	201
202 Accepted	202.php	202
203 Non-Authoritative Information	203.php	203
204 No Content	204.php	204
205 Reset Content	205.php	205
206 Partial Content	206.php	206
300 Multiple Choices	300.php	300
301 Moved Permanently	301.php	301
302 Found	302.php	302
303 See Other	303.php	303
304 Not Modified	304.php	304
307 Temporary Redirect	307.php	307
308 Permanent Redirect	308.php	308
400 Bad Request	400.php	400
401 Unauthorized	401.php	401
402 Payment Required	402.php	402
403 Forbidden	403.php	403
404 Not Found	404.php	404 (There is an option not to show 404)
405 Method Not Allowed	405.php	405
406 Not Acceptable	406.php	406
407 Proxy Authentication Required	407.php	407
408 Request Timeout	408.php	408
409 Conflict	409.php	409
410 Gone	410.php	410
411 Length Required	411.php	411
412 Precondition Failed	412.php	412
413 Payload Too Large	413.php	413
414 URI Too Long	414.php	414

Figure 4.24 File in directory with different response

the tool with the common HTTP responses found mostly it shows the correct value except for 103 (Early Hints), 418 (I'm a teapot) and 425 (Too early) that return 500 (Internal Server Error Response). Testing with the different file permission also shows that it is working correctly which shows 403 (Forbidden) when the permission is not allowed, and it should be able to run through the file within the folder. Using redirect on .htaccess could overwrite the current folder or file and show 302 Found (temporary redirect) response. The tool has some flaws, but it seems not that important so let us say that the tool is reliable enough for the testing.

4.3.2 SQLmap

For SQLmap we test it against server with known SQL injection vulnerabilities and see if SQLmap can identify said vulnerabilities correctly. We use Damn Vulnerable Web Application (DVWA) as target. DVWA is a web application design with vulnerabilities to help hackers test their skill legally. DVWA has 4 different security settings (Low, Medium, High, Imposible), the highest setting (Imposible) is meant to be vulnerability free and cannot be hacked.

SQLmap can accurately identify DVWA's vulnerabilities at all hackable levels and was able to retrive information in DVWA's databases as shown in figure below. (Figure 4.25, Figure 4.26, Figure 4.27)

Figure 4.25 SQLmap result against DVWA on Low.

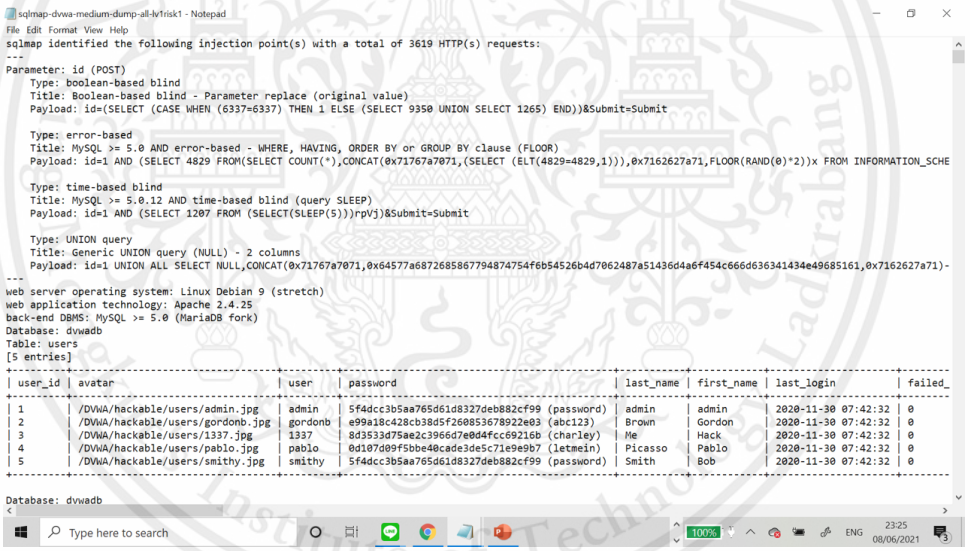

Figure 4.26 SQLmap result against DVWA on Medium

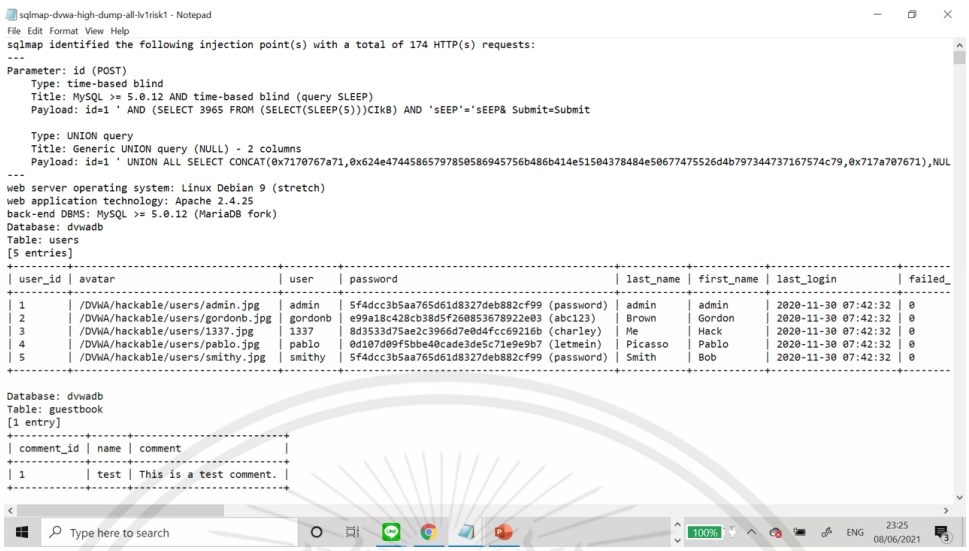

Figure 4.27 SQLmap result against DVWA on High.

4.3.3 Hydra

Hydra attacks by dictionary so the successful rate is dependent on how well the word list is. The longer of the word list could slow down the process which is the tradeoff. On the first test, using a build in password for each protocol which provide a smaller size word list with the name related to the protocol. The test conduct on three protocols which are FTP, SSH, and Telnet as on Figure 4.8. The build-in file could guess for the FTP and SSH but not for the telnet. Which could be conclude that the build-in password is not good enough for the production but good enough for testing. Another test tries on using hydra on the key-based authentication as in the Figure 4.28. The result was immediately exiting the program because of the different system.

Files Input Entries(number)	Status	Protocol	Port	User	Password	Result	Duration	CPU	Memory	Space Log Size
31	Open	FTP	21	user	pass	Succeeded	1 minute 36 seconds	26.71%	85%	698B
21	Open	SSH	22	local	ssh	Succeeded	2 minutes 0 seconds	28.24%	87%	697B
111	Open	Telnet	23	telnet	root	Failed	5 minutes 58 seconds	27.14%	86%	928B

Files Input Entries(number)	Status	Protocol	Port	User	Password	Result	Duration	CPU	Memory	Space Log Size
21	Open	SSH	22	local	ssh	ERROR	11 seconds	31.36%	86%	417B

Figure 4.28 Hydra first test case result

4.3.4 Amass

For Amass we test it against known domain name with known subdomains (cmkl.ac.th). We manually ping every subdomain on the list and get their IP address and all of them are correct. This might not cover all of the CMKL's subdomains since Amass manage to enumerate only 30 subdomains (Figure 4) but the person who manage domain names for CMKL told us that they are around 50 subdomains under CMKL (she was not sure about the exact number).

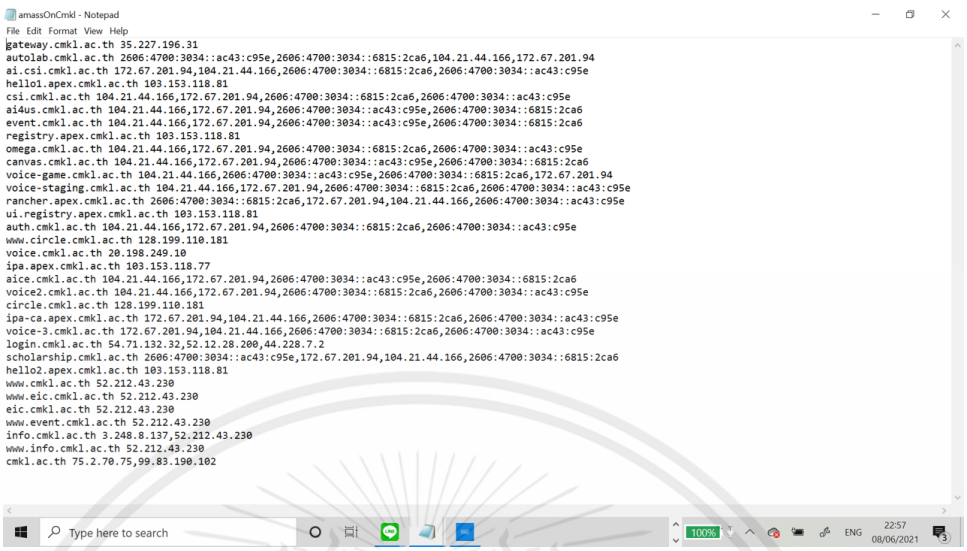

Figure 4.29 Amass result against domain cmkl.ac.th

4.3.5 Nmap

Nmap is validated by scanning the created instances to monitor and configuration. In this test we used 5 OSs to perform the validation including Ubuntu, Fedora, FreeBSD, CentOS, and Debian for cloud-based systems as shown below. When Nmap finishes the scanning the log file will be shown(figure 4-25) in detail. Including the information of the ports opened, service name and version which we could take them as a useful information. As for the other information besides host ports we can conclude from the comparison between setup instances and Nmap finding(figure 4-26). That these other four sections are less accurate or unfindable in the target's system.

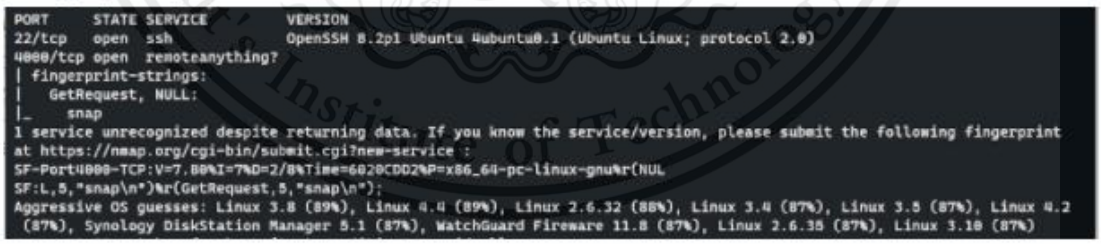

Figure 4.30 Nmap finding on Ubuntu instance

	Setup	Scan Find
OS	Ubuntu 20.04	-
Kernel	LTS Linux 5.4.0	Linux 3.8 (89%), Linux 4.4 (89%)
Fingerprinting	Yes	No
Encryption	SHA-256	-
Active Port (LISTEN only)	22, 4000	22, 4000

Figure 4.31 Comparison in Ubuntu

	Setup	Scan Find
OS	Fedora 33	-
Kernel	Linux	Linux
Fingerprinting	Yes	No
Encryption	N/A	-
Active Port (LISTEN only)	22	22

Figure 4.32 Comparison in Fedora

	Setup	Scan Find
OS	FreeBSD 12.2	FreeBSD 11.x
Kernel	Linux	Linux
Fingerprinting	Yes	Yes
Encryption	N/A	-
Active Port (LISTEN only)	22	22

Figure 4.33 Comparison in FreeBSD

	Setup	Scan Find
OS	CentOS 8.3	-
Kernel	Linux	Linux
Fingerprinting	Yes	Yes
Encryption	N/A	-
Active Port (LISTEN only)	22	22

Figure 4.34 Comparison in CentOS

	Setup	Scan Find
OS	Debian 9	-
Kernel	Linux	Linux
Fingerprinting	Yes	Yes
Encryption	N/A	-
Active Port (LISTEN only)	22, 111	22, 11

Figure 4.35 Comparison in Debian

4.3.6 Searchsploit

Searchsploit is a tool that find the vulnerability in the software programs by query the protocol name and version then it will show the information, fetched from the Searchsploit's database. Which, only a set of limited protocol could be detected in the database. Then the result will be shown along with vulnerability's name and path to website of database.


```
{
    "SEARCH": "Memcached 1.5.5",
    "DB_PATH_EXPLOIT": "/opt/exploitdb",
    "RESULTS_EXPLOIT": [
        {"Title":"Memcached 1.5.5 - 'Memcrashed ' Insufficient Control of Network Message Volume Denial of Service With Shodan API","URL":"https://www.exploit-db.com/exploits/44265"},
        {"Title":"Memcached 1.5.5 - 'Memcrashed' Insufficient Control Network Message Volume Denial of Service (1)","URL":"https://www.exploit-db.com/exploits/44264"},
        {"Title":"Memcached 1.5.5 - 'Memcrashed' Insufficient Control Network Message Volume Denial of Service (2)","URL":"https://www.exploit-db.com/exploits/44254"}
    ],
    "DB_PATH_SHELLCODE": "/opt/exploitdb",
    "RESULTS_SHELLCODE": [  ]
}
```


Figure 4.36 Get information of an outdated or unpatched of the program

4.3.7 Resource Usage

Resource usage measured by monitoring the status from Kubernetes dashboard while running the various tools on the Argo workflow as in Figure 4.37.

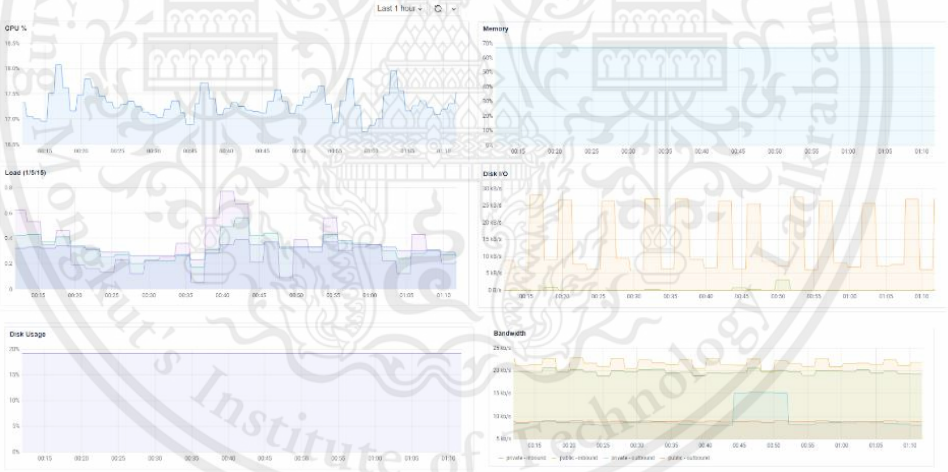

Figure 4.37 The sample of droplet resource usage when scanning the endpoint.

Different tools use different resource usage and time duration. Figure 4.37 show the VM data usage dashboard, and it was used for the measurement in Figure 4.38.

Nothing				
CPU: 20.93%				
MEMORY%: 64%				
Endpoints	**DNSMAP**	**NMAP**	**DIRB**	**XSSSNIPER**
188.166.231.190 (Juice Shop)		ID: test-tool-bwtlv	ID: test-tool-ss7ld	ID: test-tool-wgmwb
		Duration: 32s	Duration: 2 minutes 57 seco	Duration: 31s
		CPU:31.57%	CPU: 25.63%	CPU:36.84%
		Memory:69.70%	Memory: 67%	Memory:64.35%
		Space size:1.3KB	Space size: 852B	Space size: 1.4KB
159.65.14.172 (BWAPP)		ID: test-tool-c65p9	ID: test-tool-hxlff	ID: test-tool-qklrm
		Duration: 30s	Duration: 11 seconds	Duration: 9s
		CPU:27.46%	CPU:25.88%	CPU:36.84%
		MEMORY:69.70%	MEMORY:65.57%	MEMORY:64.35%
		Space size:1.7KB	Space size:756B	Space size:1.6KB
[redacted]	ID: test-tool-w8bw5	ID: test-tool-b4gtt	ID: test-tool-hbvnr	ID: test-tool-z8m8w
	Duration: 36s	Duration: 44s	Duration: 1 hour 42 minutes	Duration: 9s
	CPU: 32.92%	CPU: 24.89%	CPU: 23.43% - 27.45%	CPU:34.17%
	MEMORY: 64%	MEMORY: 66%	MEMORY: 65-67%	MEMORY:65.12%
	Space size: 0 B	Space size: 22.1 KB	Space size: 1.2 KB	Space size: 1.5 KB
[redacted]	ID: test-tool-5cqp8	ID: test-tool-s7mv5	ID: test-tool-zldsr	ID: test-tool-chcr5
	Duration: 27s	Duration: 47s	Duration: 1 hour 16 minutes	Duration: 18s
	CPU: 31.77%	CPU: 29.86%	CPU: 27.72 %	CPU:34.17%
	MEMORY: 64%	MEMORY: 64%	MEMORY: 65 %	MEMORY:65.12%
	Space size: 0 B	Space size: 2 KB	Space size: 1.8 KB	Space size:1.5KB

Figure 4.38 The sample of droplet data usage when scanning the endpoint. It is shown that some tools have various run times from minutes to an hour depending on the characteristics of the endpoints.

4.4 System Evaluation

The evaluation of the system has been done by testing the system against the real-world endpoints on the security company (section 4.4.1) and the organization endpoints (sections 4.4.2)

4.4.1 Security company endpoints

Allow permission to scan a company endpoint, CSAAS system Ip had been whitelisted to reach their service during the time window they provided. The scanning on Figure 4 was scanned on the company endpoints with the Amass, Nmap and Hydra. The following scan results apply to the company endpoint from Figure 4 in addition to the Dirb tool. The company given 2 endpoints were domain name so according to the activity diagram in Figure 4 it will run the Amass first. The scan couldn't find the subdomain, so it moves on to the Nmap, one of the result logs found in Figure 4, and in founding the protocols name SSL/HTTP-proxy, which was not supported by the current hydra, so it did not pass the parameter to hydra to process. During the given time window, Dirb was also tested, and the result log is in Figure 4.40.

Figure 4.39 Nmap result on the company endpoint. Found the only port opened is 443 running on the HA Proxy load balancer version 1.3.1, this information could be used for checking on the outdated software. It also provides the size and algorithm of the key bit that can further be tested for the outdated implementation. The OS found is Linux with the specific version that based on guessing.

Figure 4.40 Dirb result of found directories on the company endpoint mostly show http response 302 which mean the page have been moved and http 400 which mean the page exist, but something went wrong with the request e.g., invalid request syntax on client side.

4.4.2 The organization endpoints

Providing the list of 10 endpoints from the organization, a scan has been performed for all endpoints with the flow as Figure 4.41 endpoints have been blocking the Ip probing so Nmap could not run through and only 4 endpoints continue the assessment. The insights from scanning the organization endpoints are which product did they use in the system and learn more about the organization structure. Commonly in a large company we found a massive log of information such as Nmap. So, same as needed a data cleaner.

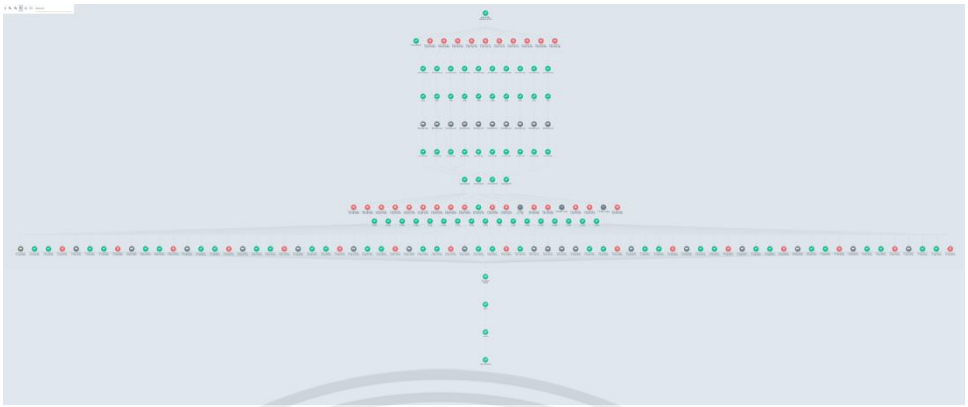

Figure 4.41 The Argo workflow scan has been run from the Argo template with the parameters of a list of the endpoints and company name.

CHAPTER 5
CONCLUSION

5.1 Introduction

In this Chapter, we first summarize the work described in this report (section 2). Then we draw several conclusions about key parts of the work undertaken in section 3. and finally, proof our system function as intended (section 4).

5.2 Summary

This is a summary of each chapter intro and summary.

Chapter 1 introduced to cyber security world and importance of this project.

Chapter 2 reviewed the state-of-the-art in security vulnerabilities and tools considered. Penetration tools were introduced, and penetration methods described. The potential for an automated security assessment system was highlighted.

Chapter 3 describes the design of the system. The separate functions of blah blah that support the requirements were then described in more detail, including blah and blah. described the implementation of the tools in our system.

Chapter 4 presented a series of tests that demonstrate the tools we choose are accurate and our system can correctly identify vulnerabilities in target systems.

Chapter 5 conclusions of this report.

Conclusions

The aim of this project was to improve the security hygiene of the treading world. We chose to focus on making the security testing process easier so it can be done more often.

We then designed and implemented a system that could:

1. Get information from the target.
2. Automatically perform security for target system
3. Find vulnerability in target system.
4. Feedback a report to company’s broker
5. Automatedly generate a report.

REFERENCES

[1] Alejandro Hernandez (2017, September 26). *Are You Trading Securely? Insights into the (In)Security of Mobile Trading Apps* [Blog Post]. Retrieved from: <https://ioactive.com/are-you-trading-securely-insights-into> [Accessed 26 September 2020]

[2] netsparker. *HIPAA Compliance Report* <https://www.netsparker.com/support/hipaa-compliance-report/#hipaa-compliance-report-sections> [Accessed 29 October 2020]

[3] OWASP. *OWASP Juice Shop* <https://owasp.org/www-project-juice-shop/> [Accessed 28 Nov 2020]

[4] KALI. *Our Most Advanced Penetration Testing Distribution, Ever.* <https://www.kali.org/> [Accessed 3 October 2020]

[5] OWASP. *OWASP WebGoat* <https://owasp.org/www-project-webgoat/> [Accessed 21 Nov 2020]

[6] RAPID7. *Understanding the reporting data model: Facts* <https://docs.rapid7.com/nexpose/understanding-the-reporting-data-model-facts/> [Accessed 10 November 2020]

[7] Purplesec**.** *Sample Vulnerability Assessment Report - Example Institute* <https://purplesec.us/wp-content/uploads/2019/12/Sample-Vulnerability-Assessment-Report-PurpleSec.pdf> [Accessed 20 October 2020]

[8] Microsoft (2020, September 21). *SQL Vulnerability Assessment helps you identify database vulnerabilities* <https://docs.microsoft.com/en-us/azure/azure-sql/database/sql-vulnerability-assessment> [Accessed 17 October 2020]

[9] Infosec Insight (2020, july 1). *13 Vulnerable Websites & Web Apps for Pen Testing and Research* <https://sectigostore.com/blog/13-vulnerable-websites-web-apps-for-pen-testing-and-research/> [Accessed 17 October 2020]

[10] nixCraft (2007, june 7). *Understanding the Linux kernel – Anatomy of the Linux kernel*

<https://www.cyberciti.biz/tips/understanding-the-linux-kernel.html> [Accessed 26 October 2020]

[11] Acunetix, *Acunetix Web Application Vulnerability Report 2020* <https://www.acunetix.com/white-papers/acunetix-web-application-vulnerability-report-2020/#methodology> [Accessed 9 October 2020]

[12] CVSS**.** *Common Vulnerability Scoring System SIG* <https://www.first.org/cvss/> [Accessed 24 October 2020]

[13] Kali Linux Tools Listing - *Offsec Services* **<**https://tools.kali.org/tools-listing> [Accessed 10 October 2020]

[14] Kulshekhar Kabra (2020, January 23), *Building Go Web Applications and Microservices Using Gin*
<https://semaphoreci.com/community/tutorials/building-go-web-applications-and-microservices-using-gin>

[15] National Cyber Security Index. *NCSI Methodology*
<NCSI :: Methodology (ega.ee)> [Accessed 16 December 2020]

[16] OWASP, *Top Ten Web Application Security Risks | OWASP*
<https://owasp.org/www-project-top-ten/> [Accessed 16 December 2020]

[17] M. Liu, B. Zhang, W. Chen and X. Zhang, "A Survey of Exploitation and Detection Methods of XSS Vulnerabilities", *IEEE Access*, vol. 7, pp. 182004-182016, 2019.

[18] Mohd Amin Mohd Yunus, Muhammad Zainulariff Brohan, Nazri Mohd Nawi, Ely Salwana Mat Surin, Nurhakimah Azwani Md Najib, Chan Wei Liang, “Review of SQL Injection : Problems and Prevention”, INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION, 2018.

[19] Jai Puneet Singh “Analysis of SQL Injection Detection Techniques”, CIISE, Concordia University, Montreal, Qu’ebec, Canada.