**EMITTER International Journal of Engineering Technology**
Vol. 10, No. 2, December 2022, pp. 246~261
DOI : 10.24003/emitter.v10i2.705

# Web Application Security Education Platform Based on OWASP API Security Project

**Muhammad Idris, Iwan Syarif, Idris Winarno**

Department of Information and Computer Engineering
Politeknik Elektronika Negeri Surabaya (PENS), Surabaya, Indonesia
Correspondence Author: idris@polibatam.ac.id

*Received August 14, 2022; Revised September 16, 2022; Accepted October 17, 2022*

**Abstract**

The trend of API-based systems in web applications in the last few years keeps steadily growing. API allows web applications to interact with external systems to enable business-to-business or system-to-system integration which leads to multiple application innovations. However, this trend also comes with a different surface of security problems that can harm not only web applications, but also mobile and IoT applications. This research proposed a web application security education platform which is focused on the OWASP API security project. This platform provides different security risks such as excessive data exposure, lack of resources and rate-limiting, mass assignment, and improper asset management which cannot be found in monolithic security learning application like DVWA, WebGoat, and Multillidae II. The development also applies several methodologies such as Capture-The-Flag (CTF) learning model, vulnerability assessment, and container virtualization. Based on our experiment, we are successfully providing 10 API vulnerability challenges to the platform with 3 different levels of severity risk rating which can be exploited using tools like Burp Suite, SQLMap, and JWTCat. In the end, based on our performance experiment, all of the containers on the platform can be deployed in approximately 16 seconds with minimum storage resource and able to serve up to 1000 concurrent users with the average throughput of 50.58 requests per second, 96.35% successful requests, and 15.94s response time.

**Keywords**: API Security, OWASP, CTF, Risk Rating, Container.

## 1. INTRODUCTION

The growth of the API-based application ecosystem continues to grow globally. Postman reports that there have been 30 million API collections and 855 million API traffic requests made by users since 2020 [1]. The use of APIs enables rapid and innovative application development. The API allows applications to interact with external systems and also can be used to develop various application platforms such as the Internet of Things (IoT), mobile applications, and web applications. Unfortunately, behind this massive growth, there are also potential threats to the API itself. API is a double-edged sword. on the one hand, APIs help in expanding the business through shared value and utility, but on the other hand, APIs pose security and privacy problems [2]. Salt security reports industry findings that have demonstrated that APIs are the dominant application attack vector today. Recorded in just 6 months, in June 2021 the overall API traffic from its customers increased by 141% while malicious traffic also grew by 348% [3].

The effort to understand and learn about security risks is also difficult to carry out considering that testing or hacking can be considered a criminal act without permission from the application owner according to ITE Law Article 31 paragraphs 1,2 and 3 [4]. In addition, there is currently no standard for specifying educational computing environments, making it impossible to share them without having to manually rebuild and redeploy most of each environment every time it is needed [5]. Therefore, popular applications such as WebGoat, Mutillidae, and DVWA are commonly used as a target for exploitation testing as well as learning media in application security education. These three applications provide several sample vulnerability case studies that users can learn and test in a legal environment. However, API and web vulnerabilities have different security risks. The API security risk report was first issued by OWASP in 2019 [6]. This report is the first and most recent of the OWASP security reports on API security risks. Thus, exploratory efforts in understanding API-based applications are still in a process that continues to develop both in terms of implementation and security aspects. Therefore, the main goal of the proposed application is to create a CTF-based environment that provides API security challenges using container virtualization to help students, teachers, security testers, and web developers in understanding the problems faced in API-based systems.

## 2. RELATED WORKS

The concept of developing application security learning systems or application security testing is not a new approach in cyber security education and research. In recent years, the implementation of this type of application has also been utilized through various security topics. In cyber security simulation and learning application, Shin S, Seto Y, Kasai Y, Ka R, Kuroki D, Toyoda S et al [7] built a learning media platform called CyExec to help cyber security learning systems with attack and defense programs. The program is then built-in virtual box and docker technology. CyExec uses traditional web technology in the attack by using WebGoat as an experimental target. Su J, Cheng M, Wang X, and Tseng S [8] proposed a scheme to create a simulation test to assess student learning outcomes online in web security subjects called the SimTI-WS scheme. The focus topic is discussed about CSRF based on WebGoat. A different approach is proposed in this research to provide an online quiz web system that works by comparing the submitted answers from users in the form of code reactions and data analysis on the server-side. Ping C, Jinshuang W, Lanjuan Y and Lin P [9] developed a teaching media application that was implemented technically through PHP and MYSQL technology to simulate various SQL injection vulnerabilities in web applications. Lehrfeld M and Guest P [10] created a vulnerable web application for learning media by adopting the Capture-The-Flag (CTF) model to assist students in conducting ethical hacking simulations that focus on web reconnaissance, password cracking, and SQL injection. Oh S, Stickney N, Hawthorne D, Mattdhews S [11] create a tool for teaching cyber security techniques using Raspberry Pi 4 called Cyber Range. This proposed learning tool utilizes Docker as container virtualization which runs all 14 vulnerabilities in DVWA as a learning object. Mansurov A [12] created a CTF-based framework for information security learning which can be virtualized using various hypervisors such as KVM, OpenVZ, and vSPhere. The proposed framework has some tasks such as local vulnerability, web vulnerability, steganography, forensic, and cryptography. In web application security, the tasks are mainly created for vulnerabilities that can be exploited using SQL, XSS, and code injection. Aziz N, Shamsuddin S, and Hassan N [13] implemented a security learning environment called KICT Secure Coding Learning Package. This proposed application consists of 3 main components of secure coding learning: SCALT, WebGoat, and specific vulnerabilities in Java, C, and C++ programming languages.

Vulnerable applications are not only utilized as learning media but can also be used as attack targets to test a new concept, model, methodology, and tools which are related to the application security. Baş Seyyar M, Çatak F, and Gül E [14] studied a web vulnerability scanning application through access log files on a server and compared the accuracy results with the model proposed in the study. In the comparison model, the target user is no longer provided by the researcher from the start, but instead utilizes a popular vulnerable application, namely DVWA. Kritikos K, Magoutis K, Papoutsakis M, and Ioannidis S [15] surveyed vulnerability assessment (VA) tools and databases for cloud-based web applications by utilizing vulnerable applications such as DVWA and WebGoat as metrics for the accuracy and capabilities of each. each VA application in scanning for vulnerabilities Priyanka A and Smruthi S [16] conducted experiments on web vulnerabilities and compared each software tool that could detect or exploit vulnerabilities in DVWA applications. Amankwah R, Chen J, Kudjo P, and Towey D [17] used WebGoat and DVWA to conduct experiments in evaluating the performance of vulnerability scanning applications, both open-source and commercial applications. Saleem S, Sheeraz M, Hanif M, and Farooq U [18] made a model with machine learning to detect attacks on web servers. In classifying the model, the dataset used is a server access log file consisting of normal access logs, SQL injection attack logs, XSS attack logs, and Denial of Service (DOS) attack logs against DVWA applications. Steiner S, de Leon D, and Jillepalli A [19] use Multillidae vulnerable web application as a study case for developing a non-least privilege security model for its DBMS database permissions. Alazmi S and De Leon D [20] benchmarked 30 web vulnerability scanners towards OWASP Web Security 2010, 2013, 2017, and 2021. To conduct their experiment in finding the effectiveness of all the vulnerability scanners, DVWA and Mutillidae are utilized as the target testing. Rangnau T, Buijtenen R, Fransen F, and Turkmen F [21] utilized WebGoat as target testing to study the integration of continuous security testing into CI/CD pipeline. Later on, this research also conducts a security testing analysis of dynamic penetration testing and fuzzing techniques by using OWASP ZAP, JMeter, and Selenium. Yang J, Tan L, Peyton J, and A Duer K [22] proposed a Security Analysis Security Testing (SAST) tool called Priv. In their final experiment of Priv, the tool is tested against the source code of WebGoat and other vulnerable web applications to help highlight their proposed model's accuracy and effectiveness. Chen P, Zhao M, Wang J, Yu H [23] introduced a DVWA-based teaching assistant system that adopts a multi-round attack defense model to organize experimental teaching to promote student's enthusiasm for learning the processes of securing web applications.

## 3. ORIGINALITY

In recent years, the research on API topics is increasing. However, in terms of security, it has not been thoroughly explored. The dominant topics in API research, in general, are still related to design and usability, all of which belong to the technological domain of classification schemes [24]. Therefore, this research tries to discuss the security aspect of API through the implementation and security analysis of the REST API-based system. The originality of the proposed application in this research also comes from the security risks scope which is not covered by some popular web applications such as DVWA, WebGoat, and Mutillidae. The security risks that will be discussed and implemented in this research are based on the OWASP API security risks 2019 which are:

- API1: Broken Object Level Authorization
- API2: Broken Authentication
- API3: Excessive Data Exposure
- API4: Lack of Resources and Rate Limiting
- API5: Broken Function Level Authorization
- API6: Mass Assignment
- API7: Security Misconfiguration
- API8: Injection,
- API9: Improper Assets Management
- API10: Insufficient Logging and Monitoring

Although there is a clear difference in terms of system architecture between the proposed platform (microservice) and existing vulnerable applications (monolithic), there are some vulnerability similarities in the OWASP API project. For example, both architectures are prone to SQL injection and command injection vulnerability. Detail comparison of vulnerability similarity is described in table 1.

**Table 1.** Security Learning Application Against OWASP API Security

| OWASP API Security | DVWA | WebGoat | Multillidae II |
|---|---|---|---|
| API1 | - | Access Control Flaws | Insecure Direct Object References |
| API2 | Weak Session IDs | Authentication Flaws, JWT Tokens and Password Reset | Authentication Bypass, Privilege Escalation, and Username Enumeration |
| API3 | - | - | - |
| API4 | - | - | - |
| API5 | - | Missing Function Level Access Control | Missing Function Level Access Control |
| API6 | - | - | - |
| API7 | - | Insecure Communication | Directory Browsing, SSL Misconfiguration, CORS Misconfiguration |
| API8 | SQL Injection, XSS Injection, Command Injection | SQL Injection, XSS Injection, Command Injection | SQL Injection, XSS Injection, Command Injection |
| API9 | - | - | - |
| API10 | Brute Force | Logging Security | Log disclosure |

Apart from the security risk scope, we also implement CTF challenges, levels, and a scoring system to the proposed application. Unlike DVWA which uses a skill-based level for its challenges, the level of each challenge in this proposed application is determined based on qualitative risk assessment methodology. The main focus of the qualitative risk assessment is the likelihood of an event rather than its statistical probability. These likelihoods are derived from analyzing the threats and vulnerabilities and then generating a qualitative value for the asset or assets that may be affected [25]. In this research, we chose OWASP risk rating methodology to provide a risk-based level challenge by determining the severity rating of each challenge that is classified into 3 levels which are medium, high, and critical. Lastly, we also proposed a container-based environment using Docker to ease the installation and configuration of requirements to run the proposed application either in a personal environment or a cyber security lab environment to accommodate multiple users simultaneously.

## 4. SYSTEM DESIGN

In achieving the goal of this research in providing an API security risk learning system, the system design is shown in Figure 1.

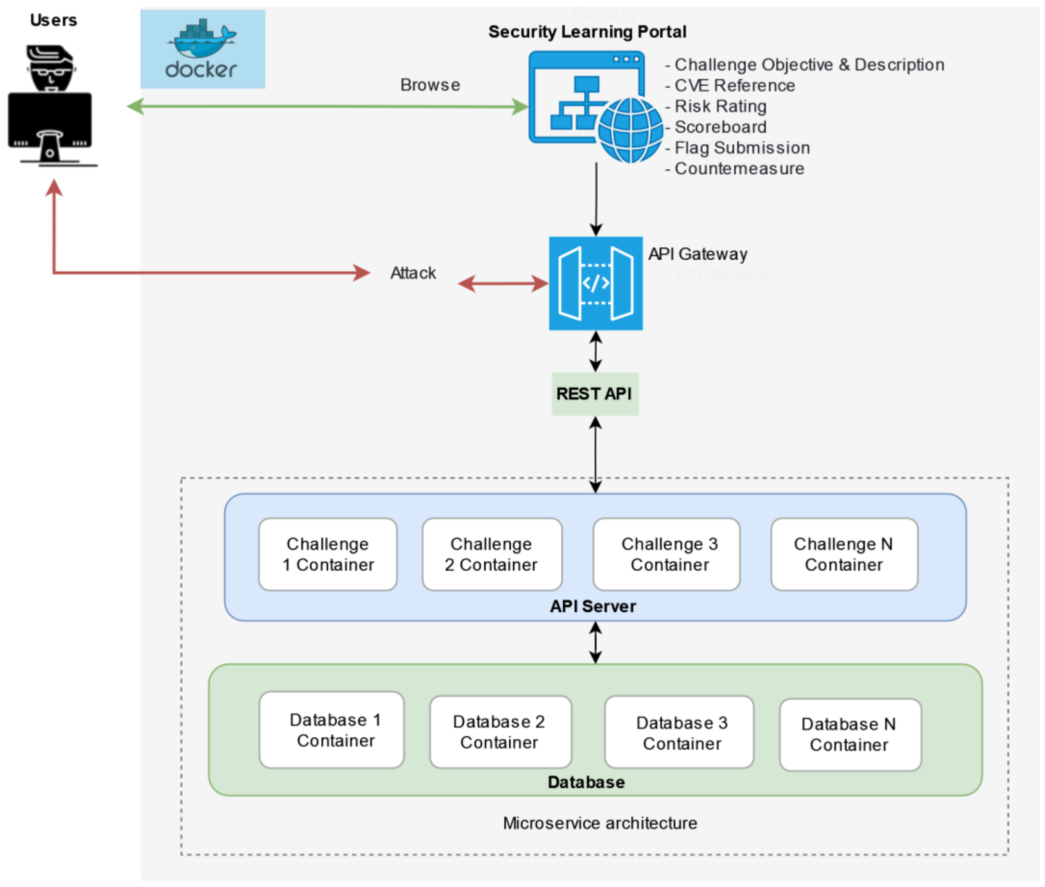

**Figure 1.** System design of the proposed platform

Docker is utilized as the container virtualization for the proposed application. With Docker container, we can maximize the simulation of each challenge vulnerability impact and provide a lightweight system at the same time. For the CTF-based learning implementation, users can browse the security learning portal to access the CTF core system features such as challenge objective and description, vulnerability references from CVE (Common Vulnerability Enumeration) and CWE (Common Weakness Enumeration) reports, challenge level based on its risk, user score, flag submission, and recommendation of countermeasures. The user and system interaction in a CTF-based system is explained in figure 2.

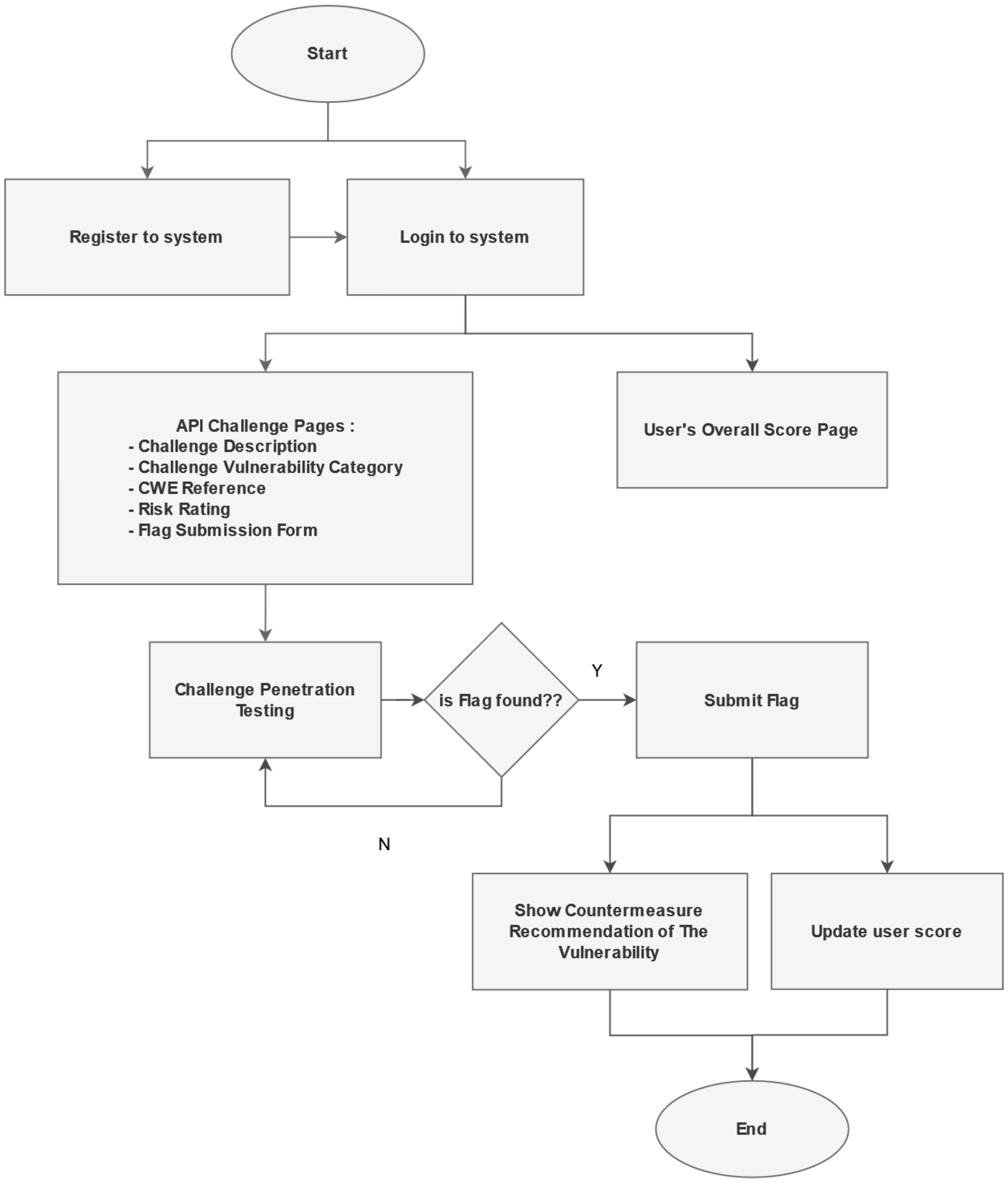

**Figure 2.** User-system interaction in the CTF learning model

Like any other software development, a challenge is created to serve a basic function of the REST API-based system. Starting from the basic function implementation, vulnerability and flag are intentionally injected into the system such as vulnerability in API endpoint parameter, response, or configuration. The proposed vulnerabilities are designed based on OWASP API security risk and adopted from vulnerability reports in CVE and CWE. The specification of the challenge design is described in table 2.

**Table 2.** Challenge design on the proposed application

| No. | API Endpoint | OWASP Risk | Challenge Objective | CVE ID | CWE ID |
|---|---|---|---|---|---|
| 1 | GET /grade/{id} | API1 | Expose another user's private grade information by tampering with the request in the API grade endpoint. | CVE-2021-44877 | CWE-285 |
| 2 | POST /login | API2 | Log in to the system by cracking insecure and weak JWT secret key | CVE-2021-40494 | CWE-287 |
| 3 | GET /course/{id} | API3 | Expose the lecturer's personally identifiable information (PII) on the course page | CVE-2019-20360 | CWE-213 |
| 4 | POST /signin | API4 API10 API2 | Use a credential stuffing attack to authenticate one of the administrator accounts | CVE-2022-24044 | CWE-307 |
| 5 | DELETE /api/ announcement /{id} | API5 | Delete announcement as non-administrator account | CVE-2019-0039 | CWE-285 |
| 6 | POST /register | API6 API2 | Gain admin role access by adding role key in JSON body request of API registration | CVE-2021-27582 | CWE-915 |
| 7 | GET /file | API7 | Exploit improper file and folder permission configuration on the API server | CVE-2020-29582 | CWE-552 |
| 8 | GET /lecturer/{id} | API8 | Gain access to the database server using vulnerable API endpoint parameter | CVE-2022-29603 | CWE-89 |
| 9 | GET /api/v1/students | API9 API3 | Find an old or beta version of API on the system to steal all of the student's PII. | CVE-2021-39905 | CWE-1059 |
| 10 | GET /api/server?info={command} | API8 API5 | Execute list home directory command in API endpoint as a non-administrator user | CVE-2021-40412 | CWE -77 |

Next, we determined the risk rating on every challenge using the OWASP risk rating methodology. OWASP risk rating is a method to measure application security risk based on the likelihood and impact that is divided into 16 different qualitative questions [26]. The likelihood is evaluated based on their threat agents and vulnerability factors, then the impact is evaluated based on technical and commercial factors. The formula that is used to determine the severity risk rating in this proposed application is:

- *Threat Agent Factors = (Skill Level + Motive + Opportunity + Size)/4*
- *Vulnerability Factors = (Ease of Discovery + Ease of Exploit + Awareness + Intrusion Detection)/4*
- *Technical Impact Factors = (Loss of Confidentiality + Loss of Integrity + Loss of Availability + Loss of Accountability)/4*
- *Business Impact Factors = (Financial Damage + Reputation Damage + Non-Compliance + Privacy Violation)/4*

After the four main factors are obtained, the likelihood and impact can be calculated using the following formula:

- *Likelihood = (Threat Agent Factors + Vulnerability Factors)/2*
- *Impact = (Technical Impact Factors + Business Impact Factors)/2*

Furthermore, by using the average value and the level of likelihood and impact in the previous step, the overall severity risk level can be determined from the matrix as shown in table 3 and table 4.

**Table 3.** Likelihood and Impact Classification Matrix

| **Level *Likelihood* dan *Impact*** | |
|---|---|
| 0 - <3 | LOW |
| 3 - <6 | MEDIUM |
| 6 - 9 | HIGH |

**Table 4.** Overall Severity Risk Level Matrix

| ***Overall Risk Severity*** | | | | |
|---|---|---|---|---|
| ***Impact*** | **HIGH** | Medium | High | Critical |
| | **MEDIUM** | Low | Medium | High |
| | **LOW** | Note | Low | Medium |
| | | **LOW** | **MEDIUM** | **HIGH** |
| | ***Likelihood*** | | | |

## 5. EXPERIMENT AND ANALYSIS

The server used in the local deployment process is using a computer with specifications of i3 12100f 3.3 GHz processor, 16 GB of DDR4 RAM, 500GB SSD, Ubuntu WSL2 on Windows 10, and Docker. To create a platform for the proposed application, at the initial stage, we implemented 3 base Docker images as described in table 5.

**Table 5.** Docker Image of Proposed Application

| Image Name | Specification | Image Source | Image Size |
|---|---|---|---|
| MicroChallenge | Apache2 and PHP8 | Dockerfile | 591.27 MB |
| MySQL | MySQL 5.7 | MySQL Official Docker Hub | 449.61 MB |
| Nginx | Nginx | Nginx Official Docker Hub | 141.52 MB |

In Docker, we can create our unique image with Dockerfile to bundle all of the technology required by the proposed application. This image called MicroChallenge will be used for API servers to run all of the specifications of challenges which are based on PHP programming language. For the database server and API gateway, both images are pulled directly from the official Docker Hub. Finally, after 3 images were created, we deployed an environment for the proposed platform. The result of this deployment is described in table 6.

**Table 6.** Docker deployment of the proposed platform

| Container Name | Docker Image | Deployment Time | Container Initial Size |
|---|---|---|---|
| Challenge1 | MicroChallenge | 14.1s | 251B |
| Challenge2 | MicroChallenge | 13.8s | 251B |
| Challenge3 | MicroChallenge | 13.6s | 251B |
| Challenge4 | MicroChallenge | 15.2s | 251B |
| Challenge5 | MicroChallenge | 14.8s | 251B |
| Challenge6 | MicroChallenge | 15.4s | 251B |
| Challenge7 | MicroChallenge | 14.4s | 251B |
| Challenge8 | MicroChallenge | 13.2s | 251B |
| Challenge9 | MicroChallenge | 13.0s | 251B |
| Challenge10 | MicroChallenge | 13.8s | 251B |
| Challenge_Portal | MicroChallenge | 16.9s | 251B |
| Challenge_Database | MySQL | 5.2s | 4B |
| Portal_Database | MySQL | 15.5s | 4B |
| API_Gateway | Nginx | 16.0s | 1.09KB |

Based on table 6 result, the container initial size of each challenge is relatively small because all of the source code was not mounted inside the container. The source code is available in the Docker host and can be accessed to each deployed container with Docker’s volume mechanism. Furthermore, the vulnerability assessment of penetration testing and risk analysis methodology based on OWASP risk rating on each challenge was executed. The result of this assessment is described in table 7.

**Table 7.** Vulnerability assessment of proposed challenges

| **Container Name** | **Penetration Testing Tool(s)** | **Vulnerable Asset** | **Exploitation Technique** | **Likelihood Level (0-9)** | **Impact Level (0-9)** |
|---|---|---|---|---|---|
| Challenge1 | Burp Suite | API Endpoint Request | API Parameter Tampering | 7.25 | 5 |
| Challenge2 | JWTCat | JSON Web Token | JWT Secret Key Cracking | 4 | 8 |
| Challenge3 | Burp Suite | API Endpoint Response | Sniffing | 7 | 7.75 |
| Challenge4 | Burp Suite | API Endpoint Request | Credential Stuffing | 5 | 8 |
| Challenge5 | Burp Suite | API Endpoint Request | API Parameter Tampering | 7.5 | 7 |
| Challenge6 | Burp Suite | API Endpoint Parameter | Privilege Escalation | 5 | 7.75 |
| Challenge7 | Burp Suite | API Server Configuration | API Parameter Tampering | 6.5 | 2 |
| Challenge8 | SQLMap | API Endpoint Parameter | SQL Injection | 6 | 8.5 |
| Challenge9 | Burp Suite | API Endpoint Response | API Parameter Tampering | 4 | 4 |
| Challenge10 | Burp Suite | API Endpoint Parameter | Command Injection | 4 | 9 |

From the results of the likelihood and impact levels obtained in table 7, we determine the overall security risk based on the OWASP risk rating matrix table as described in table 4. From 5 levels of severity risk rating, there are none of the challenges has a note or low rating. All of the challenges are at minimum have a medium severity risk rating as shown in Figure 3.

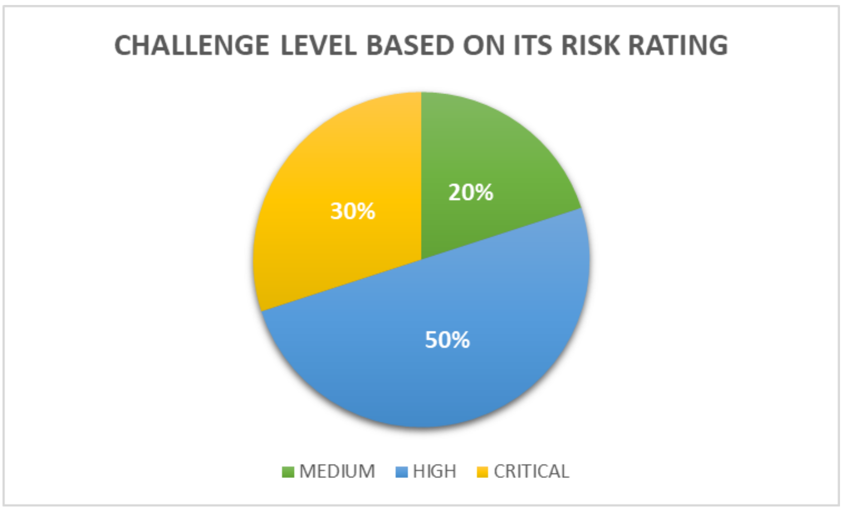

**Figure 3.** Challenge level based on its risk rating

Next, we evaluate the performance of the proposed platform using the load testing technique using K6 testing software. load test is performed on a local environment on the same device of the proposed platform. To provide real case testing, we create user-flow testing which will request 3 main API endpoints for each scenario as shown in figure 4.

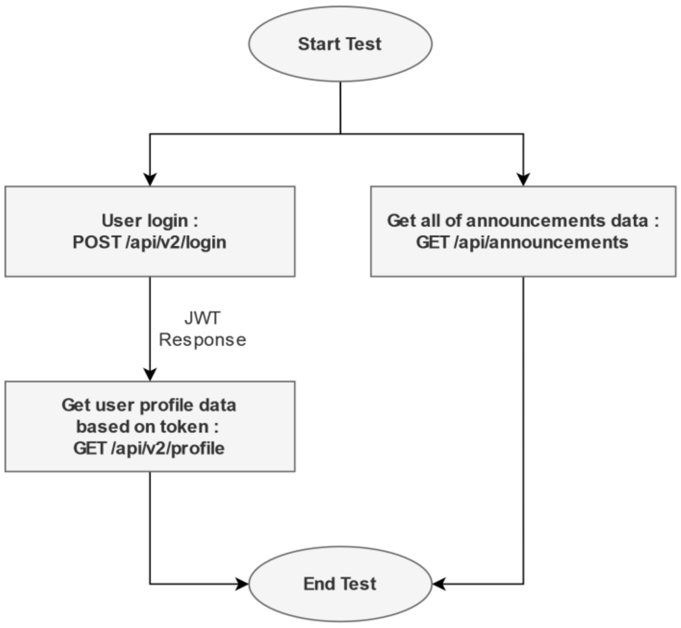

**Figure 4.** Scenario-based testing for performance evaluation

Furthermore, we perform a load test to analyze the performance of the proposed container-based challenge. The first testing starts with 100 concurrent users (CU) and gradually increased to 1000 concurrent users. With the total of 3 API endpoints as target testing as designed in figure 4, the iteration of each testing is set at 3 times of the total CU and 1s idle time for each API endpoint call. The result of this testing is described in table 8.

**Table 8. Load Testing on Container Challenge**

| Concurrent Users | Average Requests (rps) | Data Received (mB/s) | Successful Request | Failed Request | Average Response time (s) |
|---|---|---|---|---|---|
| 100 | 28.14 | 1.01 | 900 | 0 | 2.13 |
| 200 | 40.69 | 1.45 | 1800 | 0 | 3.41 |
| 300 | 48.73 | 1.74 | 2700 | 0 | 4.56 |
| 400 | 51.72 | 1.85 | 3600 | 0 | 5.77 |
| 500 | 51.05 | 1.82 | 4500 | 0 | 7.68 |
| 600 | 51.79 | 1.85 | 5400 | 0 | 9.3 |
| 700 | 50.58 | 1.81 | 6300 | 0 | 11.27 |
| 800 | 50.2 | 1.79 | 7196 | 0 | 13.8 |
| 900 | 49.7 | 1.75 | 7884 | 119 | 14.7 |
| 1000 | 50.58 | 1.76 | 8572 | 313 | 15.94 |

Based on load testing performance results as shown in Table 4, the proposed container-based challenge is capable of serving up to 800 concurrent users with approximately 7192 successful requests and 0% of error rate as shown in Figure 5.

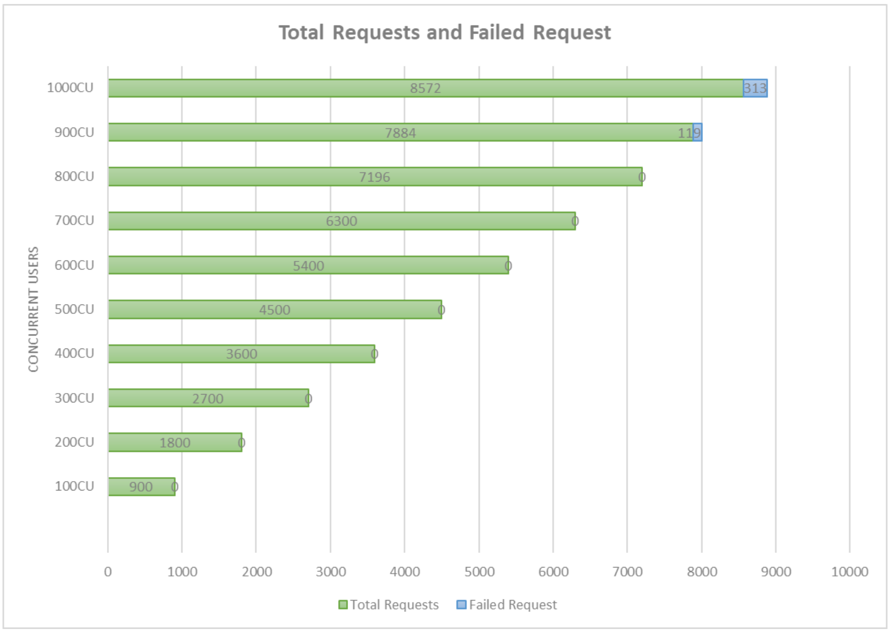

**Figure 5.** Total requests and failed requests

However, when the performance test was set to 900 and 1000 concurrent users, multiple failed requests occurred on the system with the highest error rate of 3.6% on 1000 concurrent user scenario. Finally, based on load test result as shown in Figure 6, the container-based challenge is able to handle requests up to 1000 concurrent users with average throughput of 50.58 requests per second and average response time of 15.94s.

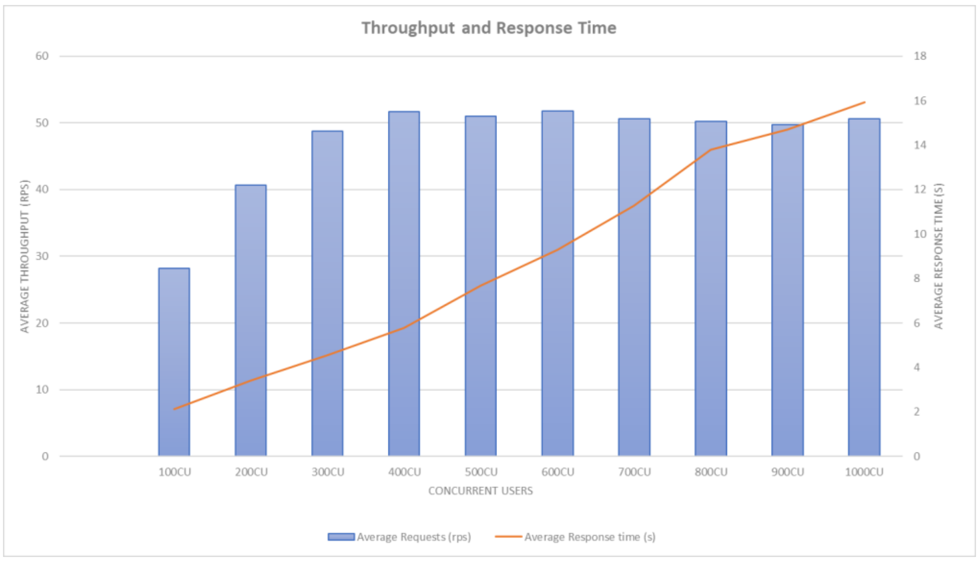

**Figure 6.** Average requests and response time

## 6. CONCLUSION

This research proposed a new platform to provide a legal and safe environment to learn API security based on OWASP API Security Project. By proposing microservice architecture in our proposed platform, we can provide different kinds of security risks such as lack of resources and rate-limiting, excessive data exposure, mass assignment, and improper assets management which do not exist in popular security learning applications like DVWA, WebGoat, and Mutillidae II. We also design and implement a security learning portal with a CTF-based model and features such as challenge, risk-based level, flag submission, and user scoring system. To provide a real case sample for the API challenges, we adopted some real-life case incidents from CVE and CWE reports. After the implementation step, a vulnerability assessment is performed against the proposed application. 10 challenges are attacked to verify the exploitability of injected vulnerability using tools such as Burp Suite, JWTCat, and SQLMap. The exploitation result also provides knowledge in determining a severity risk rating for risk-based level categorization which has the result of 5 high-level challenges, 3 critical challenges, and 2 medium challenges. Based on the performance evaluation, the container-based can serve up to 1000 concurrent users with an average throughput of 50.58 requests per second and 96.35% of successful requests and 15.94s response time. However, our recommendation is to provide this proposed platform with only 400 concurrent users which have an acceptable response time of 5.77s and 0% error rate percentage.

For future work, the number of challenges should be increased to cover more kinds of vulnerabilities in API-based systems. Improvement of the proposed platform including container-based API server, database server, and API gateway also should be optimized to achieve 100% successful requests percentage on 900 to 1000 concurrent users and lower response time.

**Acknowledgments**

We would like to thank the Politeknik Elektronika Negeri Surabaya and Politeknik Negeri Batam for the supports on this research project.

## REFERENCES

[1] **2021 State of the API Report [Internet]**, *Postman*, 2021 [cited 23 March 2022], Available from: https://www.postman.com/state-of-api/.

[2] **API Security Trends [Internet],** *Salt.security*, 2021 [cited 23 November 2021], Available from: https://salt.security/api-security-trends.

[3] Hussain F, Hussain R, Noye B, Sharieh S. **Enterprise API Security, and GDPR Compliance: Design and Implementation Perspective**. *IT Professional*, vol. 22, no. 5, pp. 81-89, 2020.

[4] **UU No. 19 Tahun 2016 [Internet]**, *Kominfo*, 2022 [cited 23 March 2022], Available from: https://web.kominfo.go.id.

[5] Conte de Leon D, Goes CE, Haney MA, Krings AW. **Adles: Specifying, deploying, and sharing hands-on cyber-exercises**. *Computers & Security*, vol. 74, pp. 12–40, 2018.

[6] **OWASP API Security - Top 10 [Internet]**, *OWASP*, 2019 [cited 23 March 2022], Available from: https://owasp.org/www-project-api-security/.

[7] Shin S, Seto Y, Kasai Y, Ka R, Kuroki D, Toyoda S et al. **Development of Training System and Practice Contents for Cybersecurity Education**. *2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI)*, pp. 172-177, 2019.

[8] Su J, Cheng M, Wang X, Tseng S. **A Scheme to Create Simulated Test Items for Facilitating the Assessment in Web Security Subject,** *Twelfth International Conference on Ubi-Media Computing (Ubi-Media)*, pp. 306-309, 2019.

[9] Ping C, Jinshuang W, Lanjuan Y, Lin P. **SQL Injection Teaching Based on SQLi-labs**. *2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE)*, pp. 191-195, 2020.

[10] Lehrfeld M, Guest P**. Building an ethical hacking site for learning and student engagement**, *SoutheastCon*, 2016, pp.1-6, 2016.

[11] Oh S, Stickney N, Hawthorne D, and Matthews S. **Teaching Web-Attacks on a Raspberry Pi Cyber Range**, *Proceedings of the 21st Annual Conference on Information Technology Education*, pp. 324-329, 2020.

[12] Mansurov A. **A CTF-Based Approach in Information Security Education: An Extracurricular Activity in Teaching Students at Altai State University, Russia**. *Modern Applied Science*, 2016.

[13] Aziz N, Shamsuddin S, Hassan N. **Inculcating Secure Coding for beginners**. *2016 International Conference on Informatics and Computing (ICIC)*, pp. 164-168, 2016.

[14] Baş Seyyar M, Çatak F, Gül E. **Detection of attack-targeted scans from the Apache HTTP Server access logs**. *Applied Computing and Informatics*, vol. 14, no. 1, pp. 28-36. 2018.

[15] Kritikos K, Magoutis K, Papoutsakis M, Ioannidis S. **A survey on vulnerability assessment tools and databases for cloud-based web applications**. *Array*, vol. 3-4, pp. 100011, 2019.

[16] Priyanka A, Smruthi S. **Web Application Vulnerabilities: Exploitation and Prevention**. *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, pp. 729-734, 2020.

[17] Amankwah R, Chen J, Kudjo P, Towey D. **An empirical comparison of commercial and open-source web vulnerability scanners**. *Software: Practice and Experience*, vol. 50, no. 9, pp. 1842-1857, 2020.

[18] Saleem S, Sheeraz M, Hanif M, Farooq U. **Web Server Attack Detection using Machine Learning**. *2020 International Conference on Cyber Warfare and Security (ICCWS)*, pp. 1-7. 2020.

[19] Steiner S, de Leon D, Jillepalli A. **Hardening web applications using a least privilege DBMS access model**. *Proceedings of the Fifth Cybersecurity Symposium*, Article 4, pp. 1–6, 2018.

[20] Alazmi S, De Leon D. **A Systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners**. *IEEE Access*, vol. 10, pp. 33200-33219, 2022.

[21] Rangnau T, Buijtenen R, Fransen F, Turkmen F. **Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines**. *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, pp. 145-154, 2020.

[22] Yang J, Tan L, Peyton J, A Duer K. **Towards Better Utilizing Static Application Security Testing**. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 51-60, 2019.

[23] Chen P, Zhao M, Wang J, Yu H. **Exploration and practice of the experiment teaching of web application security course**. *2019 10th International Conference on Information Technology in Medicine and Education (ITME)*. 2019.

[24] Ofoeda J, Boateng R, Effah J. **Application Programming Interface (API) Research**. *International Journal of Enterprise Information Systems*, vol. 15, no. 3, pp. 76-95, 2019.

[25] Kuzminykh I, Ghita B, Sokolov V, Bakhshi T. **Information security risk assessment.** *Encyclopedia*, vol. 1, no. 3, pp. 602–17, 2021.

[26] **OWASP Risk Rating Methodology [Internet]**. *OWASP*, 2015 [cited 25 March 2022]. Available from: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology.