



ADMAS UNIVERSITY
SCHOOL OF POST GRADUATE STUDIES
DEPARTMENT OF COMPUTER SCIENCE

COMPARISION OF SYNCHRONOUS AND ASYNCHRONOUS
PROGRAMMING FOR DESIGNING NETWORKED WEB APPLICATION
SECURITY FRAMEWORK

A Thesis Submitted to Department of Computer Science of
Admas University in Partial Fulfillment of the Requirements for
the Degree of
Masters of Science in Computer Science

By

Seife Akalu

Advisor: Mulugeta A (Asst prof)

Date: July 19, 2021

THESIS APPROVAL SHEET
ADMAS UNIVERSITY
SCHOOL OF POSTGRADUATE STUDIES

DECLARATION

I, Seife Akalu, the under signed, declare that this thesis entitled: Designing Novel Web Application Security Policy Framework is my original work. I have undertaken the research work independently with the guidance and support of the research supervisor. This study has not been submitted for any degree or diploma program in this or any other institutions and that all sources of materials used for the thesis has been duly acknowledged.

Name of Student

Signature

Date

This is to certify that the thesis entitled: [Designing Novel Web Application Security Policy Framework] submitted in partial fulfillment of the requirements for the degree of Masters of [Science in Computer Science] of the Postgraduate Studies, Admas University and is a record of original research carried out by [Seife Akalu] [PGMGC /7508/18], under my supervision, and no part of the thesis has been submitted for any other degree or diploma. The assistance and help received during the course of this investigation have been duly acknowledged. Therefore, I recommend it to be accepted as fulfilling the thesis requirements.

Name of Supervisor

Signature

Date

Name of Co-Supervisor (if any)

Signature

Date

Certificate of Approval

This is to certify that the thesis prepared by (Seife Akalu), entitled “Designing Novel Web Application Security Framework” and submitted in partial fulfillment of the requirements for the Degree of Masters of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signature of Board of Examiner`s:

_____	_____	_____
External examiner	Signature	Date
_____	_____	_____
Internal examiner	Signature	Date
_____	_____	_____
Dean, SGS	Signature	Date

Thesis Title: _____, Admas University, Ethiopia.

Submitted by (Seife Akalu): _____

Signature: _____ Date: _____

DECLARATION

I, the undersigned, declare that the thesis comprises my own work. In compliance with internationally accepted practices. I have acknowledged and referenced all materials used in this work. I understand that non-adherence to the principles of academic honesty and integrity, misrepresentation/ fabrication of any idea/data/fact/source will constitute sufficient ground for disciplinary action by the University and can also evoke penal action from the sources which have not been properly cited or acknowledged.

Signature: _____

Name of the student: _____

Date: _____

ACKNOWLEDGMENT

Foremost, I would like to thank GOD for giving me strength and health I need to better proceed with my research and then I would like to express my sincere gratitude to my advisor Asst Prof. Mulugeta.A for the continuous support of my M.Sc. study and research. Other than my advisor, I must thank DireDawa University Department of Computer Science every staff and former Department Head Dr. Girum D. I wouldn't have made it this far if it weren't for you, Thank you DireDawa University.

Besides my advisor, I would like to thank the rest of my thesis committee; Finally, I must express my very profound gratitude to my family and my Father Akalu Nureye for his support, motivation and guidance. This accomplishment would not have been possible without him. Thank you.

Contents

CHAPTER ONE	1
BACKGROUND	1
1.1 Introduction.....	1
1.2 Statement of the Problem.....	2
1.3 Objective of the Study.....	3
1.3.1 General Objective	3
1.3.2 Specific Objective.....	3
1.4 Scope of the research	4
1.5 Significance of the Research.....	4
1.6 Conceptual framework.....	4
CHAPTER TWO	6
LITRATURE REVIEW	6
2.1 Introduction.....	6
2.1.1 Website Security Risks	6
2.2 Related Work	10
2.2.1 The PHP Programmer’s Guide to Secure Code.....	10
2.2.2 A New Framework of Security Vulnerabilities Detection in PHP Web Application	11
2.2.3 Security Assessment of PHP Web Applications from SQL Injection Attacks.....	11
2.2.4 Securing web applications with secure coding practices and integrity verification...	11
2.2.5 Securing Web Applications	12
2.2.6 Principled and Practical Web Application Security	13
2.2.7 Security in Web Applications and the Implementation of a Ticket Handling System	13
CHAPTER THREE	15
RESEARCH METHODOLOGY	15
3.1 Introduction.....	15
3.1.1 Architecture of the Proposed Approach.....	15
3.1.2 Design of the Proposed Approach.....	16
3.1.3 Proposed approach	16
3.2 Methodology Used.....	17
3.3 Tool Used.....	17
3.3.1 Back-end Library Used	17

3.3.3	Vulnerability Scanner	18
3.3.4	Performance Measurement Tool Used.....	18
3.4	Security Policies.....	19
3.4.1	Basic Web Application Security Policy with JavaScript	19
CHAPTER FOUR.....		26
IMPLEMENTATION AND ANALYSIS		26
4.1	Introduction.....	26
4.1	Developed Secure Web Application	26
4.2	Test Application Vulnerability.....	28
4.2.1	API Vulnerability Checklist.....	28
4.2.2	Routs of the Application	29
4.2.3	Login Attempts and Hash Password	29
4.2.4	Setting Valid Cookie After login attempts.....	30
4.2.5	Rate Limiter for Brute-force attack.....	31
4.2.6	Rate Limiter for DOS Attack	32
4.2.7	AES Encryption	33
4.2.8	General Report From Vooki	33
4.2.9	Generated CSRF Token for every Page Request	34
4.2.10	SQL Injection With Malicious Code.....	34
4.3	Why Framework is needed for JavaScript	36
4.3.1	PHP Code	37
4.3.2	JavaScript Code	38
4.3.3	Setting Request	38
4.3.3.1	Set Request URL and Port Number	39
4.3.3.2	Set Number of Trades	39
4.3.3.3	Set Result Tree.....	40
4.3.4	Generated Performance report.....	40
4.3.4.1	PHP Script Performance with 40 Trades	40
4.3.4.2	JavaScript Performance with 40 Trades	41
4.3.4.3	Summary of Result	42
4.3.5	Performance Comparison of SQL Injection of JavaScript and PHP	42
4.3.5.1	PHP Code.....	43

4.3.5.2	JavaScript Code.....	43
4.3.5.3	Setting HTTP Request	44
4.3.5.4	Setting Traded Group.....	45
4.3.5.5	Showing Performance Result.....	46
CHAPTER FIVE		48
CONCLUSTION AND RECOMMENDATION		48
5.2	Conclusion	48
5.3	Recommendation	48
Reference.....		49

List of Tables

Table 1 Test Checklist..... 28

Table 2 Test Cases..... 37

List of Figures

Figure 1 Conceptual framework of the study	5
Figure 2 Average Web Vulnerabilities by OWASP	6
Figure 3 Same-Origin Protocol Workflow	12
Figure 4 Architecture	15
Figure 5 Security Design.....	16
Figure 6 Login page.....	26
Figure 7 Create Stream for Student and Instructor	27
Figure 8 Student Class Course.....	27
Figure 9 Instructor Class Course	28
Figure 10 Login API Testing	29
Figure 11 Login Success result	30
Figure 12 Setting Cookie	30
Figure 13 Rate Limiter Limit.....	31
Figure 14 Error code After Limit Expires.....	31
Figure 15 Rate Limiter DOS limit.....	32
Figure 16 Response message if DOS Expires.....	32
Figure 17 Encryption of Instructor Phone number	33
Figure 18 Vulnerability Test Report	33
Figure 19 List of Vulnerability Found	34
Figure 20 CSRF Token.....	34
Figure 21 SQL injection Malicious Code.....	35
Figure 22 Login Success.....	35
Figure 23 SQL Injection Error	36
Figure 24 Request URL.....	39
Figure 25 Number of trades.....	39
Figure 26 Result tree.....	40
Figure 27 PHP Report from HTML.....	40
Figure 28 Load time of PHP Script.....	41
Figure 29 JavaScript Report from HTML	41
Figure 30 JavaScript Load time	42
Figure 31 PHP HTTP Request	44
Figure 32 JavaScript HTTP Request.....	45
Figure 33 PHP Traded Group	45
Figure 34 JavaScript Traded Group.....	46
Figure 35 PHP Response time	46
Figure 36 JavaScript Response time.....	47

Acronyms

AES.....	Advanced Encryption Standard
API.....	Application Programming Interface
CSRF.....	Cross-site Request Forgery
DOS.....	Denial of Service
HTTP	Hypertext Transfer Protocol
OS.....	Operating System
OWASP.....	Open Web Application Security Project
PHP.....	Hypertext Preprocessor
SQL.....	Structured Query Language
XSS.....	Cross site scripting

Abstract

Developer's primary choice is the web for designing and deploying applications because of its cross platform and cross browser compatibility. However, security concern it is often ignored in the development stages of the web applications. Moreover, developers are more focused to produce working features for the applications in the rapid development, than providing security for the code and often do not practices secure coding. Therefore, countless web applications are launched with security vulnerabilities which manifest later in their life cycle. Integrating security features should be part of the development process for web applications to prevent web application attacks. If developers or software engineers are not practicing secure coding and having an integrity verification system in place, it is difficult to prevent attacks. After studying about web application vulnerability and attack types, the study addressed the main research question "do we need a new security framework for Web Application development which that mostly rely on networking." By studding past secure frameworks and doing new Secure JavaScript framework and testing the framework for vulnerabilities by using manual and tool based approach, this research has answered this question by comparing performance of past programming language with new JavaScript framework. Because both programming language PHP and JavaScript is used for applications that are highly dependent on network which that frequently send and receive data from remote server, using Jmeter performance measurement tool, by sending sample test request to remote server and evaluating response time and latency, performance of PHP secure framework from past studies and new secure JavaScript framework has been shown. Depending on the performance result, this research concluded by claiming JavaScript is fast because its Asynchronous nature also mentioned this language is very suitable for networked application and needs secure framework. Finally, this study recommended further open-source security framework is needed for this programming language.

Keyword: Security framework, Vooki, Jmeter, JavaScript, PHP

CHAPTER ONE

BACKGROUND

1.1 Introduction

A web application refers to any computer program that carries out specific functions via a web browser that is used to run the application. Developer's primary choice is the web for designing and deploying applications because of its cross platform and cross browser compatibility. The Web platform is a complex ecosystem composed of a large number of components and technologies, including HTTP protocol, web server and server-side application development technologies (e.g., NodeJS, PHP, and ASP), web browser. Vulnerability is a weakness which can be exploited by a threat actor, such as an attacker, to cross privilege boundaries within a computer system. And web application security deals specifically with the security surrounding websites, web applications and web services such as APIs. API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Today, web applications have progressed more and they are now a stable for businesses of all sizes. They allow for seamless day to day business operations and communication. While they are extremely beneficial, these web applications are like open doors to any business. If they are not made secure, cybercriminals can manipulate the application to provide them any data that they are looking for, including sensitive business information. However, information security and privacy issues are not always taken into consideration properly when web applications are developed. Thus, most web applications are victims to cyber-attacks and can be prone to giving out valuable information. The most vulnerable applications are internet based business and applications that deal with a user's personal information such as credit card information and insurance records. According to Mohamed Al-Ibrahim's Journal, A great deal of attention has been given to network-level security, such as port scanning, and great achievements have been accomplished at this level as well. However, it was found that about 75% of attacks were targeted to application-level, such as web servers and web applications (Al-Ibrahim, 2014). Also according to Xiaowei Li and Yuan Xue's Survey, a high percentage of web applications deployed on the Internet are exposed to security vulnerabilities. And according to a report by the Web Application Security Consortium, about

49% of the web applications being reviewed contain vulnerabilities of high risk level and more than 13% of the websites can be compromised completely automatically, and a recent report reveals that over 80% of the websites on the Internet have had at least one serious vulnerability (Jun Zhu *, 2013). Therefore, it is becoming increasingly important to develop web applications that have proper security policies, measures in place and protect the applications that have already been deployed.

1.2 Statement of the Problem

Web applications can be attacked on both the client side and the server side, and many computer security problems are caused by software vulnerabilities, software loopholes that can be easily exploited by attackers and result in data and financial loss as well as inconvenience to customers. There are many causes for such vulnerabilities, but the most common ones are flaws or loopholes introduced by developers during program design and development, and a good common example for this is data validation and sanitization, client side data cannot be trusted and should always be validated before usage. Without practicing secure coding, design pattern, security policies, and having an integrity verification system in place, web applications are launched with security vulnerabilities. According to Journal of Advanced Research (Xiaowei Li, 2014), a number of vulnerabilities can be addressed with relatively straightforward coding practices, referred to as secure programming. Bad programming practice is typically difficult to detect through manual code review. This captures the risks that come from vulnerabilities like buffer overflows, input sanitization and validation mistake, format string vulnerabilities and various other code-level mistakes, where the solution was to properly design the application with security practice in mind.

This thesis will analyze and study these problems to answer the following research questions:

1. How can developers implement secure Back-end application which that responds well to client application?
2. How can developers protect front-end applications business logic from reverse engineering?
3. How developers can implement secure user authentication and authorization?

4. How can developers protect user data when transmitted to the network and when it is kept to the database?
5. How the new security framework performs when compared with other frameworks?

For these questions to be answered, after developing Asynchronous security framework, this study first develops Simple student registration web application and evaluates the web applications in terms of vulnerability metrics.

1.3 Objective of the Study

Research objectives describe concisely what the research is trying to achieve. They summarize the accomplishments a researcher wishes to achieve through the project and provides direction to the study.

1.3.1 General Objective

The general objective of the study is to design security framework for a web application that mostly rely on networking.

1.3.2 Specific Objective

Specific objectives of the research are:-

1. To conduct review of the literature related to the problem.
2. To design strong web Back-end API and Front-end Code Security Practices.
3. To Design Web Front-end Application reverse engineering security Policies.
4. To design secure and efficient authentication and authorization technique
5. To design data protection when it is transmitted to the network and when it is kept in to the database.
6. To Test vulnerability of developed framework.
7. To compare performance of the designed framework with other existing designed framework and show the gap.

1.4 Scope of the research

This study uses **JavaScript** and will focus on preparing secure **Coding Practice**, strong web API and Web Application security policy.

And the study's focus area is only web application that executes on the browser and which that mostly rely on networked application, and will focus on common web application vulnerability and protection and these are: Injection Flaws Protection, Broken Authentication and Session Management, Cross-Site Scripting (XSS) Protection, Sensitive data storage and protection, Cross site request forgery (CSRF), Brute-force and DOS/DDOS Attack Protection, Front-end-Application Reverse Engineering Protection. And will not cover advanced security including front-end computer and mobile application.

1.5 Significance of the Research

The proposed secure code practice and web application security policy will help to prevent many web application cyber-attacks from happening because it removes the vulnerabilities many exploits rely on. Here are some significant and beneficial achievements that the proposed web security policies are expected to provide:

- Provide common web security algorithms and design concepts for software developers to reduce web application vulnerabilities.
- Provide policies which developer can use to protect front-end application from reverse engineering.
- Prepare design of algorithms and flowcharts for secure authentication and authorization which developer can use during the development of an application.
- Prepare design and algorithms of secure data transmission and storage.

1.6 Conceptual framework

For illustrating this research better, the below research model is developed. The below image (Figure 1) of conceptual framework shows, the steps that will be followed and the tasks that will be done for carrying out this research.

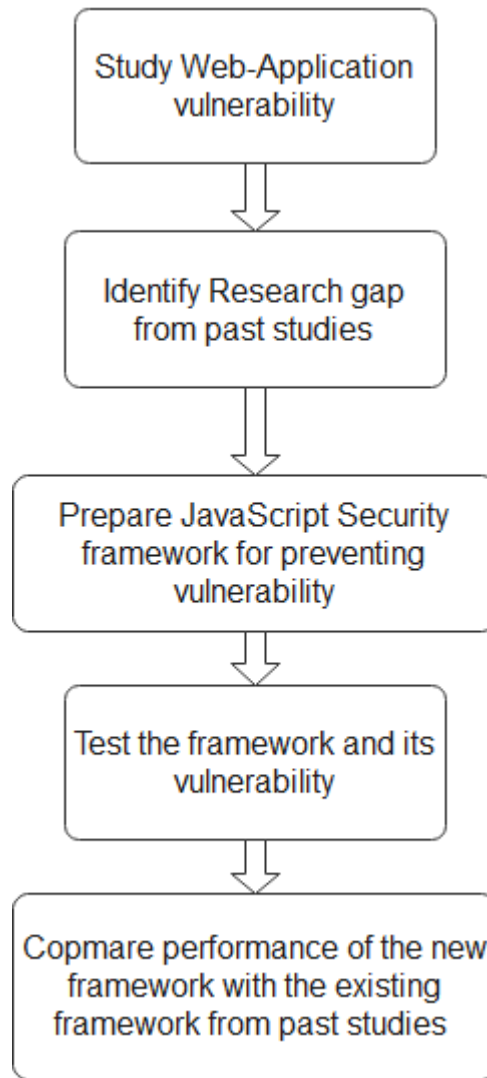


Figure 1 Conceptual framework of the study

The above image shows, after studying about Web application security, and identifying the research gap from existing framework, this research prepares JavaScript code and test the new framework if it prevents attack and then finally, this study compares performance of the new Asynchronous JavaScript security framework with other frameworks from the existing studies.

CHAPTER TWO

LITRATURE REVIEW

2.1 Introduction

This chapter contains a review of existing literature on web application security. Specifically, the chapter describes relevant theories, definition of relevant terminologies of the topic.

2.1.1 Website Security Risks

Organizations have been solely dependent upon security measure at the perimeter of networks, such as firewalls and intrusion detection in order to protect IT infrastructures. Nevertheless, now that numerous attacks are been geared towards security flaws in web design and web application, such as injection flaws, the traditional way of network security may not be adequate to safeguard web and web application and users . Ten security risks have also been identified by Open Web Application Security Project (OWASP) as the most critical security risks associated with web applications (Vincent Appiah, 2017).

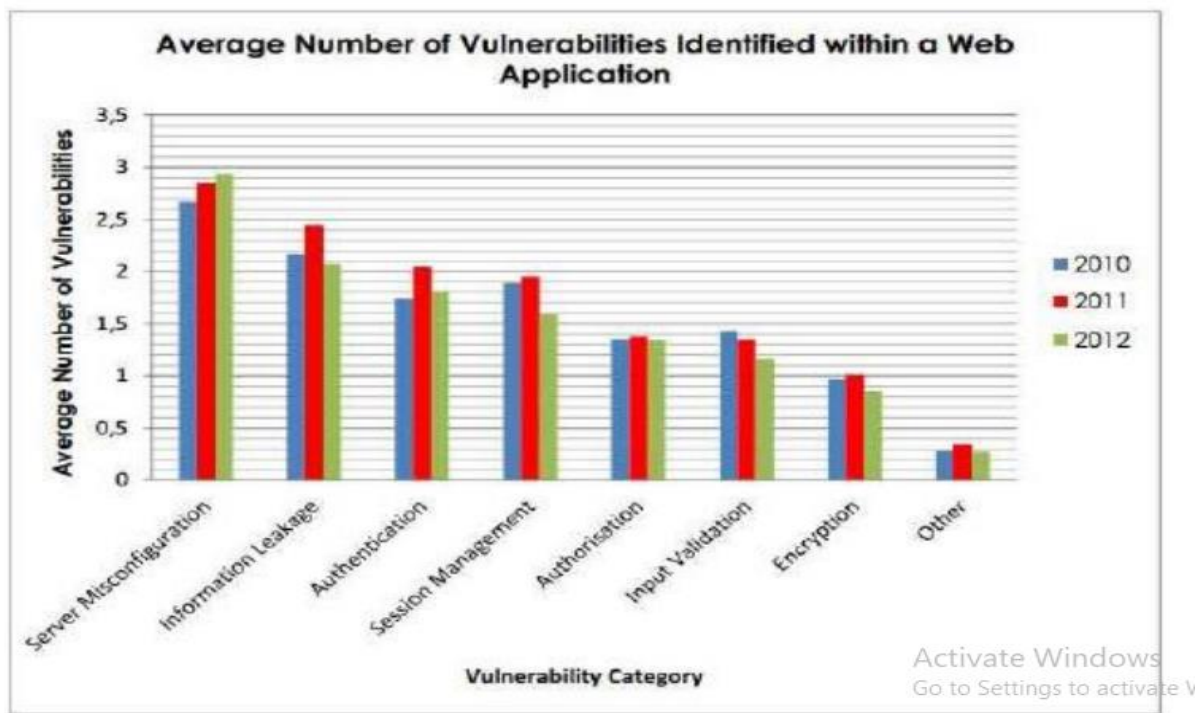


Figure 2 Average Web Vulnerabilities by OWASP

These risks are known to be common forms of attacks. (Vincent Appiah, 2017) The top 10 risks as published by OWASP are:-

- Injection flaws
- Broken authentication and session management.
- Cross site scripting.
- Insecure direct object references.
- Security misconfiguration.
- Sensitive data exposure.
- Missing level access control.
- Cross site request forgery (CSRF).
- Using components with known vulnerabilities.
- Invalidated redirects and forwards.

Injection Flaws: - institute explains that injection flaws occur when an unexpected data is sent by a malicious client. Injection flaws allow an attacker to inject code into the vulnerable computer system. If the injected code is executed, the effect can be disastrous. Aside from the stealing information, injection attacks can cause denial of service or multiplication of worms in a system. Injection attacks include SQL injection, OS injections and LDAP injections. Injection flaws occur when a user input is not properly filtered for string escape characters that are often embedded in SQL statements.

Broken Authentication and Session Management: - Broken Authentication and Session Management: This stems from the fact that flaws exist in session management implementations in web applications. Misconfigurations such as storage of passwords in plain texts or weak encryption of user credentials can lead to this form of attack. According to OWASP, flaws in the implementation of password management, logout mechanism, and timeout, remember me, forgot my password etc. can also lead to broken authentication and session management attacks.

Cross-Site Scripting (XSS):- This is a type of vulnerability in which malicious code injected by a client is executed by the web application. The execution is made possible because the web application is unable to properly filter input properly. This can lead to stealing of cookies, website defacement and session hijacking. XSS is amongst the most common vulnerabilities of web applications. There are three main types of XSS and these are; Stored XSS, Reflected XX and DOM based XSS.

Insecure Direct Object References: - This is where unauthenticated clients are given access to restricted resources such as directories and configuration files. An example is a situation where a directory or a password file that should be available to only administrators on network is exposed to other users on the network. The absence of access control check can often result in unauthorized access to such resources through manipulation of URL parameters.

Sensitive data exposure: - Sometimes sensitive data is left unprotected on web applications. These can be stolen or modified by attackers and used to gain access or perform unauthorized transactions. Using weak encryption schemes can also result in sensitive data exposure. Attackers can use brute-force to obtain the plain text. Also sensitive data can be used to exploit the web application or find other exploitable vulnerabilities on the web application.

Missing Level Access Control: - This occurs when users are not properly authenticated but given access to restricted resources. A web application must be able to limit and control the access to resources. If the application is unable to do this, then attackers can leverage this to gain access to restricted resources and even modify data on the server. This might affect the integrity of the data. There should be security checks to ensure that a user is properly authenticated and given the proper access rights especially if several users with different roles are exist on the web application.

Cross-Site Request Forgery (CSRF): - This is a type of attack where unauthorized HTTP requests are sent from a user's browser to a web application in which the user is currently logged on. In contrast to XSS, CSRF exploits the trust that a site has in a user's browser. Because there is trust, the web application is forced to execute these requests.

Using Components with Known Vulnerabilities: - Applications, with known vulnerabilities are likely to be compromised. If such applications are compromised, an attacker might gain full access to the network and this will affect confidentiality.

Invalidated redirects and forwards: - This is due to improper validation / invalidation of user data. Attackers can leverage this to redirect victims to malicious webpages as well. Also forwards can be used to access restricted pages. This can affect confidentiality of data.

Buffer overflows: - This is the situation where data being written by a program to a buffer is more than the capacity of the buffer. As a result the extra data flows to the adjacent memory locations. For example if a program allocates 20 bytes to a memory buffer and attempts are made to store 25 bytes, the extra 5 bytes will flood to the adjacent buffer and this might cause the program to crash. If a data in that adjacent space it might be overwritten. Buffer overflows can lead to the crashing of a program (denial of service) or insertion of a remote shell which can be used to execute arbitrary codes.

Denial of Service: - This is an attack that renders an application or network unable to function properly. This is usually performed by sending several requests to the application. If the number of requests is more than it can handle, the application hangs and users will not be able to use the service. Buffer overflow attacks can also cause denial of service by flooding the memory with data.

A distributed denial of service: - is used to describe the situation where large numbers of computers are used to cause denial of service. Denial of service attacks can take several forms which include:

- Buffer overflow
- Smurf attack
- Tear drop attack

Buffer overflow attacks are usually performed by sending data which is larger than the allocated memory buffer. As a result the extra bytes flood to adjacent buffers and the program crashes.

Smurf attack: - involves the attacker sending packets to a receiving machine. The request is then sent to all hosts on the network using the broadcast address. The packet then sent to the

address indicated in the packet headers. This is usually the address of the target address (IP spoofing). Because this is a broadcast, all the hosts which received the request also send their response to the same address. If the packets are overwhelmingly large, then the target address is unable to receive all other incoming traffic.

The tear drop attack involves: - sending large packet data to the target machine. The Internet Protocol (IP) unable to handle reassembly of the packet fragments due to a confusing offset value eventually causes the system to crash.

2.2 Related Work

Studies have been done regarding web application security policy frameworks. The studies properly addressed different type of attacks and mitigation plans some of the most common attacks covered with the studies are SQL injection attacks, cross-site scripting (XSS), and cross-site request forgery and briefly presented prevention techniques most of them did the framework with PHP programming language. However, technology is changing rapidly; much has changed today in the technology world researches does not cover latest technologies. If we take JavaScript as a Back-end instead of PHP, JavaScript will make the development time of an application fast and also because of its asynchronous and real-time data processing today, developer's primary choice is JavaScript for Back-end programming. Additionally, if we take Reverse-engineering protection, latest front-end JavaScript frameworks and libraries have built-in Obfuscation algorithms which make our front-end application harder to trace. The general problem here is there is lack of studies regarding latest programming language like JavaScript and do not have framework that covers all common Web vulnerabilities in one.

Some of the more prominent candidates for similar work are:

2.2.1 The PHP Programmer's Guide to Secure Code

(Clarinsson, 2005) The study focuses on showing how to write a secure code in PHP in a way that hacker cannot exploit it. This study also aims to get information about what common security vulnerability a PHP script could contain. After discussing about PHP and Web application security threats, like Cross-Site Request Forgeries, SQL Injections, HTTP requests, Session threats, then the study presents secure coding practice techniques, source codes to properly protect PHP web application from attack.

2.2.2 A New Framework of Security Vulnerabilities Detection in PHP Web Application

(Jingling Zhao, 2015) The study targets to provide developers' programming techniques and safety awareness, the study mentioned that there are many kinds of web application security flaws loopholes and vulnerabilities hiding in the program. So the study mentioned that it is very important to improve their reliability and security. The study uses PHP programming language and this new security framework of Security Vulnerabilities Detection in PHP Web Application aims at detecting PHP security vulnerabilities. After the study discussed about SQL Injection and Cross-Site Scripting, it shows algorithms of and provides proper way of implementing secure PHP script. And finally the study prepared test cases each test case consists of a BAD program that includes vulnerability, and an OK program which patches the vulnerability.

2.2.3 Security Assessment of PHP Web Applications from SQL Injection Attacks

(Atiqur Rahman, 2015) This study discussed about SQL Injection and proposes a solution for proper way of implementation to prevent this problem and also by incorporating an SQL injection tool. This SQL Injection tool is able to inject vulnerabilities into an application that are both attackable and realistic, meaning that it is possible to use the injector to create a test application whose vulnerabilities are chosen by that evaluator. A vulnerability injection tool can also be used to estimate the number of vulnerabilities that are left to correct in an application after it has already been tested.

2.2.4 Securing web applications with secure coding practices and integrity verification

(Anis, 2018) The study focuses on how web application client sides can be protected by practicing secure coding and integrity verification. And the development of security policies for web applications component shows how secure coding practices, when implemented properly, can provide security to web applications and includes the development of an integrity verification module that prevents code tampering during runtime of web applications. To secure the client side against third party attacks, the study propose some security policies as guidelines to make web applications secure against SQL injection attacks, cross-site scripting (XSS) attacks and resource alterations. The policies are derived from secure coding practices presented in the industry. The proposed approach for integrating security in web applications has two components. Because SQL injection and XSS attacks are caused by mishandling user inputs, the

study proposes input sanitization as a first security policy for prevention. After input sanitization, the next policy proposed for ensuring security is output validation. The study strongly mentioned data from the client should be checked and verified before usage. To ensure that unwanted malware are not served to web applications through servers, every time a third party resource is added to the web application, Hash string containing the sha256 cryptographic hash value script that is to be requested is added to the HTML element to confirm its integrity. The other security policy for preventing cross-site scripting is Content Security, for preventing this; the study proposes creating whitelists which the browser can choose whether to execute or to be blocked. Additionally, to ensure that the security code on the client side is not tampered with during runtime, the study proposes having a signed result of the code calculated on the server, randomizing the signing procedure with different challenges, and verifying the signed code on the client and the server end.

2.2.5 Securing Web Applications

(Pelizzi, 2016) The study focuses on targeting two of the most common vulnerabilities, namely, cross-site scripting (XSS) and cross-site request forgery (CSRF). The option proposed for preventing cross-site request forgery is on the browser, jCSRF-script is responsible for obtaining the authentication token, and supplying it together with every request originating from this page. By comparing the domain of the current page and the domain of a request, this script can distinguish between same-origin and cross-origin requests (Figure 1), and if different site must be communicated the study suggested token must be exchanged between different origins or sites

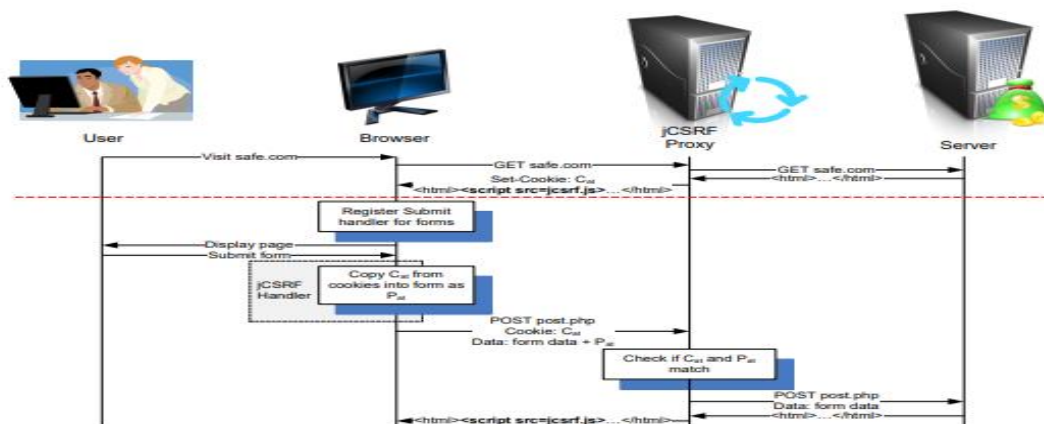


Figure 3 Same-Origin Protocol Workflow

Finally the study shows implementation of the above architecture by devised a new web application development paradigm where security policies are clearly separated from the application logic, and where users retain control of their data.

2.2.6 Principled and Practical Web Application Security

(Stefan, 2015) The study presents approach to protecting sensitive user data even when application code is buggy and malicious. The key ideas of this work are to separate the security and privacy concerns of an application from its functionality, and to use language-level information flow control (IFC) to enforce policies throughout the application codebase. The aim of this study is at once to design practical systems that can be easily adopted by average developers, and to leverage formal semantics that rule out large classes of design error. To address this challenge, this dissertation presents two systems—Hails and COWL—which respectively address the security issues web applications face on the server and in the browser. The study presents Hails as a server-side web framework that separates the security and privacy of an application from its functionality by following a new paradigm called model – policy – view - controller (MPVC). In the MPVC model, developers specify security policies in a single place, using a declarative policy specification language. Hails then enforces these policies across all application. COWL is a JavaScript confinement system that extends the browser security model with IFC for Origins and the Same-Origin Policy, postMessage and Cross-Origin Resource Sharing (CORS). For cross-origin resource Under COWL, the browser treats a page exactly like a legacy browser does unless the page executes a COWL API operation. After showing Hails Security policy, finally the study shows implementation detail of COWL in Firefox 31.

2.2.7 Security in Web Applications and the Implementation of a Ticket Handling System

(Forsman, 2014) The study focuses on finding common problem areas in web application security and how to make those areas less vulnerable to attacks. After defining security practices, the study implements ticket handling system without any security loopholes. The study addressed SQL injections and Cross-Site scripting attacks. And for preventing SQL injection from happening, the study mentioned data validation, filtering out characters or strings known to cause problems, filtering out anything except characters known to not cause problems, and Separate structure (SQL) and data. In addition to this, for protecting SQL Injecting the study

addressed another way of avoiding problems, which is, to restrict access to the minimal needed. For instance, a search application does not usually need write access to the database tables containing user account information. The study also presented protection for cross-site scripting (XSS) and cross-site request forgery with two methods. One method is to store the detail information on the server and just leave a small session identifier on the client system so you can make sure that a client uses the correct data stored on the server. Also instead of having built-in protection in the runtime environment you can make the operating system enforce some protection for the applications by providing access control by providing to access to different users. After preparing security policies and best practices the study implements Ticket handling system and test system vulnerabilities.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Introduction

Research methodology is the specific procedures or techniques used to identify, select, process, and analyze information about a topic.

3.1.1 Architecture of the Proposed Approach

Three-tier architecture is a well-established software application architecture that organizes applications into three logical and physical computing tiers: the presentation tier, or user interface; the application tier, where data is processed; and the data tier, where the data associated with the application is stored and managed. The security design and framework is going to be focused on the application tier, the application tier is going to hold all the necessary security algorithms, techniques and holds generally the new security framework. (Figure 4) shows the architecture that holds the entire application framework.

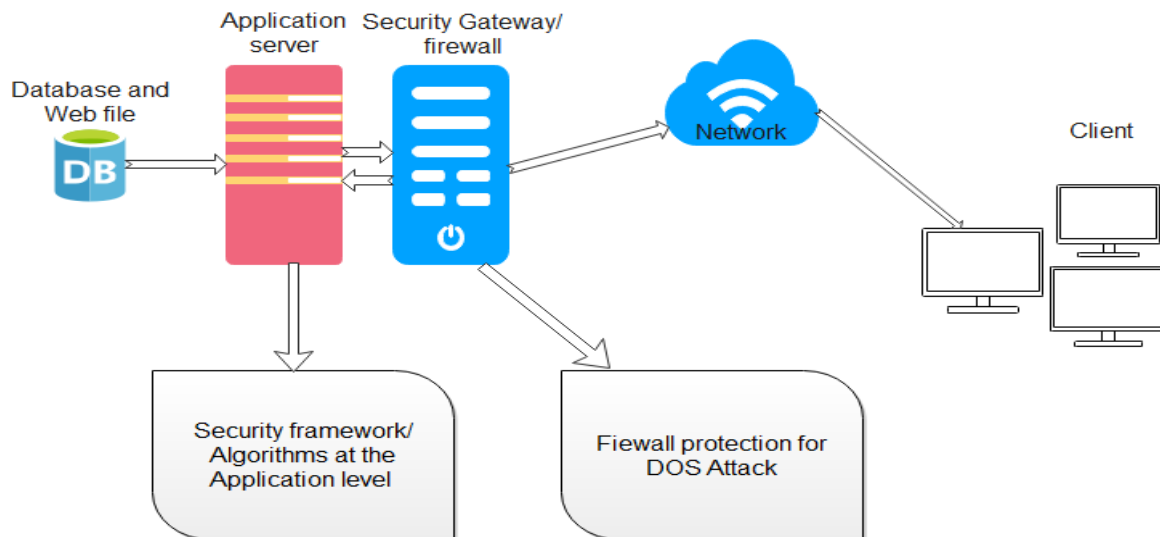


Figure 4 Architecture

3.1.2 Design of the Proposed Approach

The middle tier at the architecture design is the application layer, in three-tier architecture, the application model, view and controller is placed on it. Regarding this, all application level security protection is done at this layer.

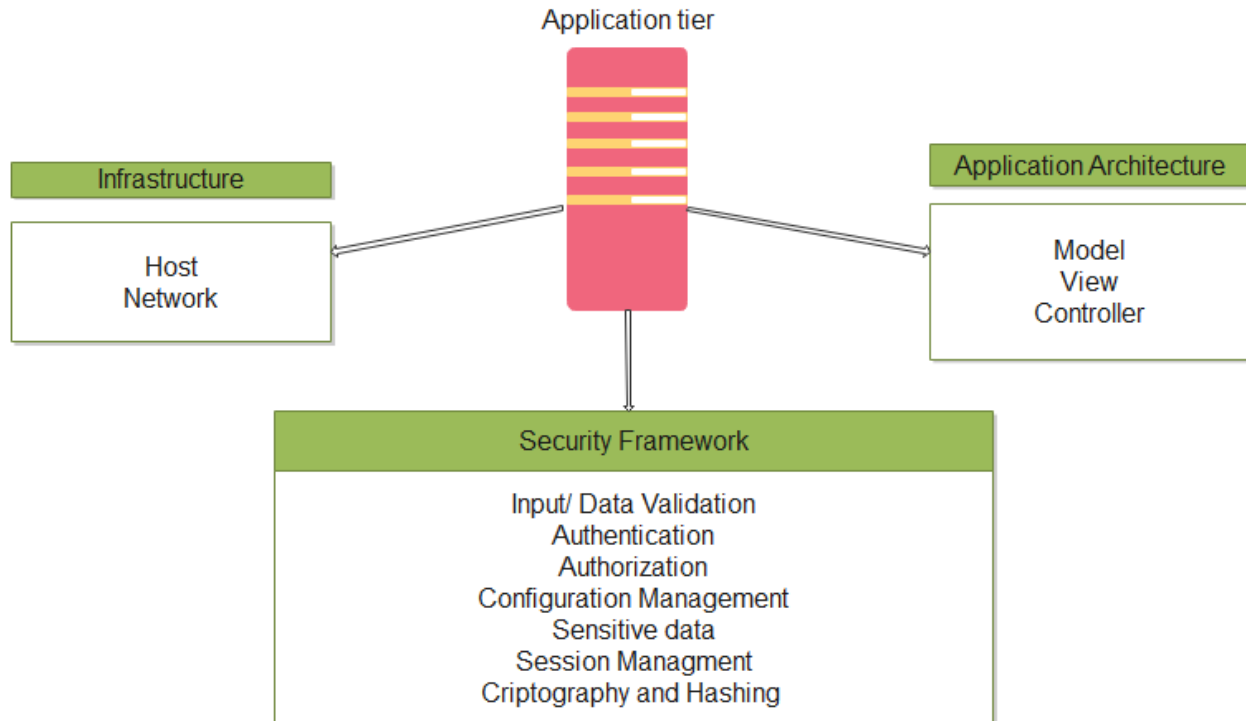


Figure 5 Security Design

3.1.3 Proposed approach

The proposed approach for integrating security in web applications has two components in it. For developing project without using other existing framework, design security policy. The security policies work alongside each other to provide security for the client side code and protect it against SQL injection, XSS attacks, Brute force attack, broken authentication, authorization, resource alteration attacks and other application level attacks. For making the designed framework easy, algorithm and flowcharts of sample code techniques will be designed.

3.2 Methodology Used

Methodology section explains about method used to showcase the outcome and to answer the question do we need framework for asynchronous programming for networked application like JavaScript because past studies have shown and developed a framework for synchronous programming example could be PHP.

3.2.1 Experimental Methodology

Experimental methods introduce erogeneity, allowing researchers to draw conclusions about the effects of an event or a program by using experiment method to evaluate the outcome. So in this study, it uses sample Vulnerability test and performance test experimental methodology to answer the question do we need to have asynchronous programming language security framework.

3.3 Tool Used

Tool used section presents programming language used for the implantation of secured web application and it also presents vulnerability scanning techniques and tools used.

3.3.1 Back-end Library Used

According to Journal (**Soomro, June 2017**) , in 2021, when talking about back-end, because most developer's primary choice is Node.js, this study will use sample source code of Node.js. And Node.js was chosen because of the following reasons:

- Easy to learn
- Easy for scalability
- Cross-platform compatibility
- Because it is asynchronous, light and fast

3.3.2 Front-end Library Used

According to React's official Docs (**Facebook, 2021**), for Client side web application reverse engineering protection, when you build the application for deployment, react application automatically Obfuscate when a project is developed with React front-end, and during production

build, it automatically Obfuscate the front end application so it is harder to trace and do the reverse engineering.

3.3.3 Vulnerability Scanner

Vooki web application vulnerability scanner tool is used because it is free web application vulnerability scanner. Vooki is a user-friendly tool that you can easily scan any web application and find the vulnerabilities. Vooki includes Web Application Scanner, Rest API Scanner, and reporting section. According to official Vooki's documentation (VegaBird, 2021), Vooki – Web Application Scanner can help to find the following attacks:

- SQL Injection
- Command Injection
- Header Injection
- Cross site scripting
- Sensitive Data Exposure and more

3.3.4 Performance Measurement Tool Used

JMeter had been used for performance testing and competition of JavaScript and PHP. According to Apache JMeter official website (Apache, 2021) , The Apache JMeter™ application is open source software, a 100% pure Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications and web API's.

Ability to load and performance test many different applications/server/protocol types:

- Web - HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, ...)
- SOAP / REST Webservices
- FTP
- Database via JDBC
- LDAP
- Message-oriented middleware (MOM) via JMS
- Mail - SMTP(S), POP3(S) and IMAP(S)

- Native commands or shell scripts
- TCP
- Java Objects

3.4 Security Policies

The first component in this study's approach is made up of several security policies that work toward preventing attacks on web applications. These policies are derived from secure coding practices and can be defined as guidelines that should be followed to protect web applications. Policies target areas are both front-end and back-end of application. The policies deal with how data must be kept in the database, how data must be transmitted across the network, how authentication and authorization must be done, how data on web applications are used and how vulnerable the applications are to the client or even an attacker. The policies might propose using JavaScript programming language security framework, input sanitization and output validation, principle of least privilege, authentication, authorization and content security.

3.4.1 Basic Web Application Security Policy with JavaScript

This section presents web application security policy which can be used during the development of any web application. This practice can be used when using existing programming framework or even in the core programming development. According to OWASP (OWASP, 2020) the following are most common security risk.

1. Reverse-engineering Protection with Code Obfuscation

According to (Savio Antony Sebastian, 2016 IEEE) survey, Obfuscation, in software technology, is the deliberate act of creating an obfuscated code which is difficult for humans to understand. And there is a need of protection during the program's execution itself, where critical code is executed and some confidential data is accessed. Even when the execution is taking place, one requires protecting the code and data from malicious intents, such as dynamic analysis and tampering. Obfuscation also prevents manual inspection of program internals by renaming variables and functions, and breaking down structures and renaming function names, variables and class names which this will make the application harder to trace for reverse

engineering this minimizes the risk for front-end client application from being reverse-engineered.

Ways in which Obfuscation essential:

- Code obfuscation promotes intensified security by preventing code modifications or ‘application hijacking’ (the insertion of malicious code).
- Software developers may also utilize obfuscation techniques to hide application flaws and vulnerabilities, or guard intellectual property.
- Code obfuscation will defend against malicious code and implementation modifications to a program and for reverse engineering an attacker must first know and understand the software before they can make specified modifications. The complicated front-end application makes the application harder to read.

Today, most front-end development frameworks have Obfuscation included. For example JavaScript front-end applications have Obfuscation when the application is built for deployment.

2. Distributed Denial of Service and Denial of Service Protection

Such an attack can be prevented using the Web Application Firewall (WAF). In terms of the throttle can be used to block the request. Most, DDOS attack should be blocked at the transport layer and at the application layer by limiting incoming requests. At the application level this study uses rate limiter to block any unusual requests.

3. HTTPS for Secure Data Transmission

If your site is processing sensitive information then make sure to deploy your site to HTTPS to safeguard its sensitive information. HTTPS avoids third-party traffic eavesdropping. And the data will be transmitted securely.

4. Broken Authentication Protection/Secure Authentication

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities temporarily or permanently (OWASP, 2020).

Proper way for authenticating user:

Step 1: when user register, generating Hash algorithm and store the Hash Password to the database

```
function generateHash(password) {  
  const hashingSecret = "PasswordSecretKey";  
  
  const hashedStr = crypto  
    .createHmac("sha256", hashingSecret)  
    .update(password)  
    .digest("hex");  
  
  return hashedStr;  
}
```

Step 2: When user tries logging in to the application, generating Hash Function and Compare it with the database password technique used.

```
exports.login = (req, res) => {  
  // validate request  
  if (!req.body) res.status(400).send({ message: "Content can not be empty!" });  
  const admin = new Admin(req.body);  
  admin.password = generateHash(req.body.password);  
  Admin.login(admin, (err, result) => {  
    //login query  
  });  
});
```

Step 3: for successful login attempt, session information storage to a server which stores its unique session id to a browser used. And then, upon successful login attempt, store logged information to the server as a session. Creating a session at the server will create automatic unique identifier to the browser as a cookie.

```
app.use(  
  session({
```

```

    secret: "334ajnbbsd",
    resave: true,
    name: "expressSessionId",
    saveUninitialized: false,
    rolling: true,
    cookie: { httpOnly: true, maxAge: 86400000 },
    store: new MemoryStore({
      checkPeriod: 86400000, // prune expired entries every 24h
    }),
  })
);

```

```

if (result.Logged) {
    req.session.authenticated = true;
    req.session.Logged = true;

}

```

Step 4: after user information is kept to the server as a session; the technique uses HTTP only cookie to the browser that is for only transmitting Cookie only on HTTP request. The advantage here is client side cookie cannot be accessed with client JavaScript code. And set a secret key, expiration date for the session and cookie. And finally set expire time of the cookie for automatically logging out user from the application.

```

    cookie: { httpOnly: true, maxAge: 86400000 },

```

Step 5: when every time a user request for other secured page access at the front-end application, sending HTTP only cookie to the server as a request header is used.

```

{ withCredentials: true }

```

Step 6: after client application sends session ID HTTP only cookie to the server, before doing any operation, for every request, checks if user is logged with specified session id.

```

if (req.session.Logged) {

```

```
// do the operation if user is logged to the application  
}
```

5. Brute-force Protection

The most obvious way to block brute-force attacks is to simply lock out accounts after a defined number of incorrect password attempts. Block an IP after a number of failed attempts, would stop the majority of attacks - block a username after a number of failed attempts on that username (say 5-10). Or the user would be sent an email to unlock their account and reset their password.

```
const loginLimiter = limiter({  
  windowMs: 2 * 60 * 1000,  
  max: 4,  
  message: {  
    code: 429,  
    message: "Toomany login requests",  
  },  
});
```

6. DOS Attack Protection

For blocking DOS attack at the application level, API request limiter had been used. For every API requester's IP Address, maximum allowed request per time had been set.

```
const DOSLimiter = limiter({  
  windowMs: 2 * 60 * 1000,  
  max: 2,  
  message: {  
    code: 429,  
    message: "Too many requests are not allowed",  
  },  
});
```

7. Cross-Site Request Forgery Protection

For preventing CSRF for every page request generate a token and provide a function the middleware will run to read the token from the request for validation.

Step 1. : For every page load, request token from the server and save the CSRF token at the front end.

```
app.get('/form', csrfProtection, function (req, res) {  
  // pass the csrfToken to the view  
  res.send({csrfToken: req.csrfToken() })  
})
```

Step 2. When requesting POST/PUT/DELETE request to the server, sending the new token to the server and verifying it for valid token.

```
{  
  "xsrftoken": csrfTokenState,  
  withCredentials: true  
}
```

8. Injection Flaws for SQL injection and Cross-Site Scripting Protection

To protect SQL injection or Cross-Site Scripting from happening, the method used is always to validate user request data before executing.

Option 1: to use escape SQL and other programming Query from the query statement.

```
connection.escape(query);  
mysql.escape(query);
```

Option 2: For SQL Injection, you can map values in the array to placeholders in the same order as they are passed

```
connection.query("SELECT * FROM user WHERE username = ? AND password = ?", [  
  req.body.dob,  
  req.body.account_number  
], function(error, results){});
```

9. Encryption

For securely storing information to the database Encryption the Advanced Encryption Standard (AES) is the algorithm trusted as the standard by the U.S. government and many other organizations.

Although it is extremely efficient in 128-bit form, AES encryption also uses keys of 192 and 256 bits for heavy-duty encryption AES is considered resistant to all attacks, with the exception of brute-force attacks, which attempt to decipher messages using all possible combinations in the 128-, 192- or 256-bit cipher. Still, security experts believe that AES will eventually become the standard for encrypting data in the private sector.

```
const crypto = require('crypto');

const algorithm = 'aes-256-ctr';
const secretKey = 'vOVH6sdmpNWjRRlQcC7rdxs01lwHzfr3';
const iv = crypto.randomBytes(16);
```

```
const encrypt = (text) => {

  const cipher = crypto.createCipheriv(algorithm, secretKey, iv);

  const encrypted = Buffer.concat([cipher.update(text), cipher.final()]);

  return {
    iv: iv.toString('hex'),
    content: encrypted.toString('hex')
  };
};
```

```
const decrypt = (hash) => {

  const decipher = crypto.createDecipheriv(algorithm, secretKey, Buffer.from(hash.iv, 'hex'));

  const decrpyted = Buffer.concat([decipher.update(Buffer.from(hash.content, 'hex')), decipher.final()]);

  return decrpyted.toString();
};
```

CHAPTER FOUR

IMPLEMENTATION AND ANALYSIS

4.1 Introduction

The goal of this chapter is to showcase the experimental approach this study undertook for the implantation of secure student registration system and test security vulnerability of the system with tool and manual based approach. It also presents the performance of the new Framework comparing with other existing PHP frameworks. The main criteria of the experiment were to make the attacks as practical as possible and following security framework developed to implement student registration.

4.1 Developed Secure Web Application

Following secure coding practice and framework developed sample secure web application was developed. The below images shows various system screens:

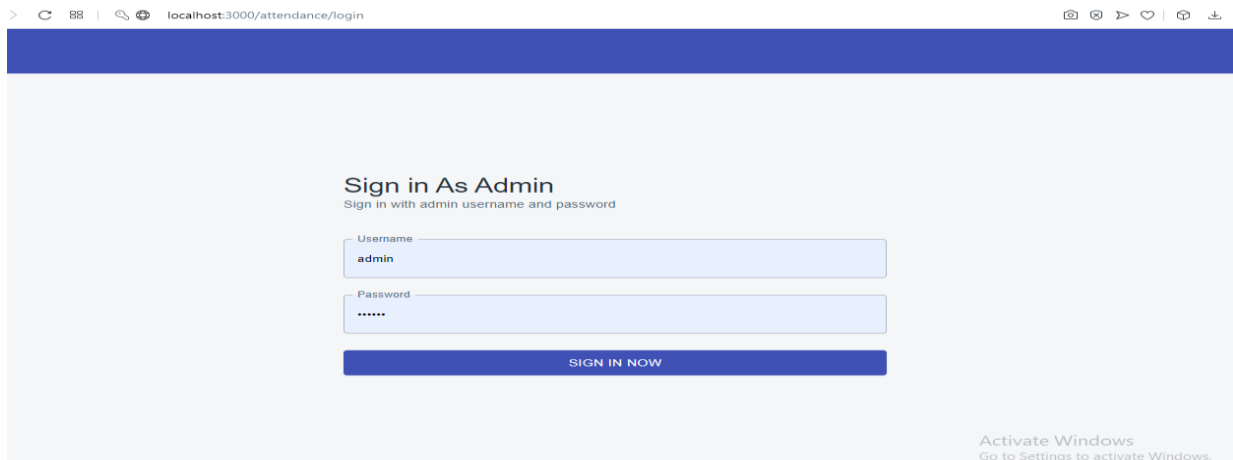


Figure 6 Login page

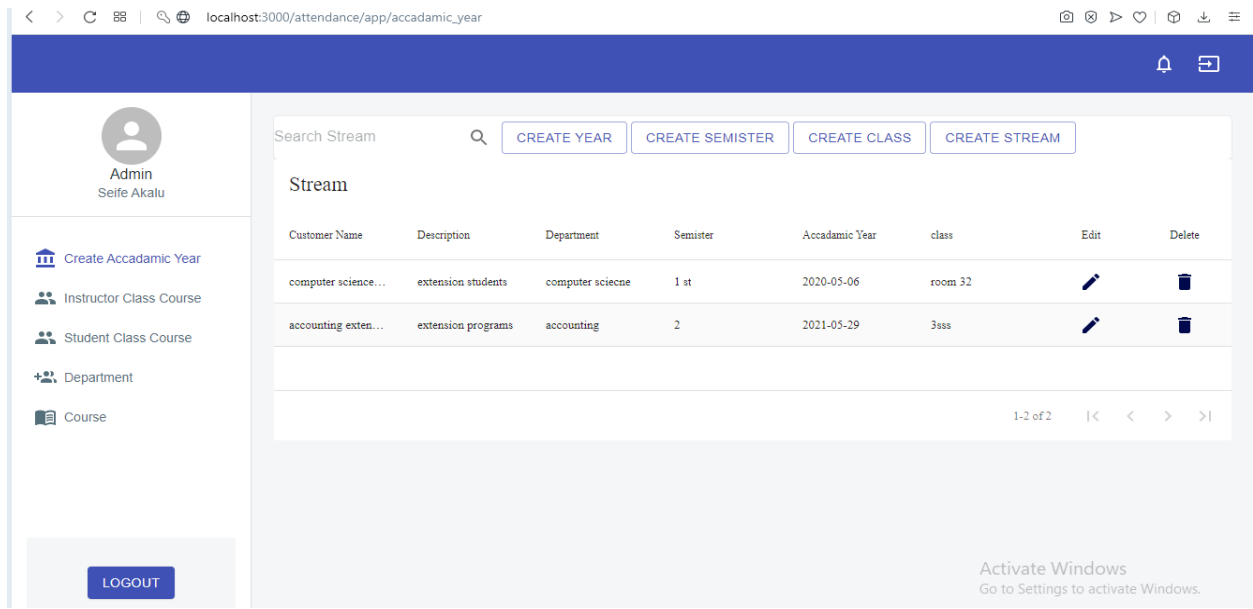


Figure 7 Create Stream for Student and Instructor

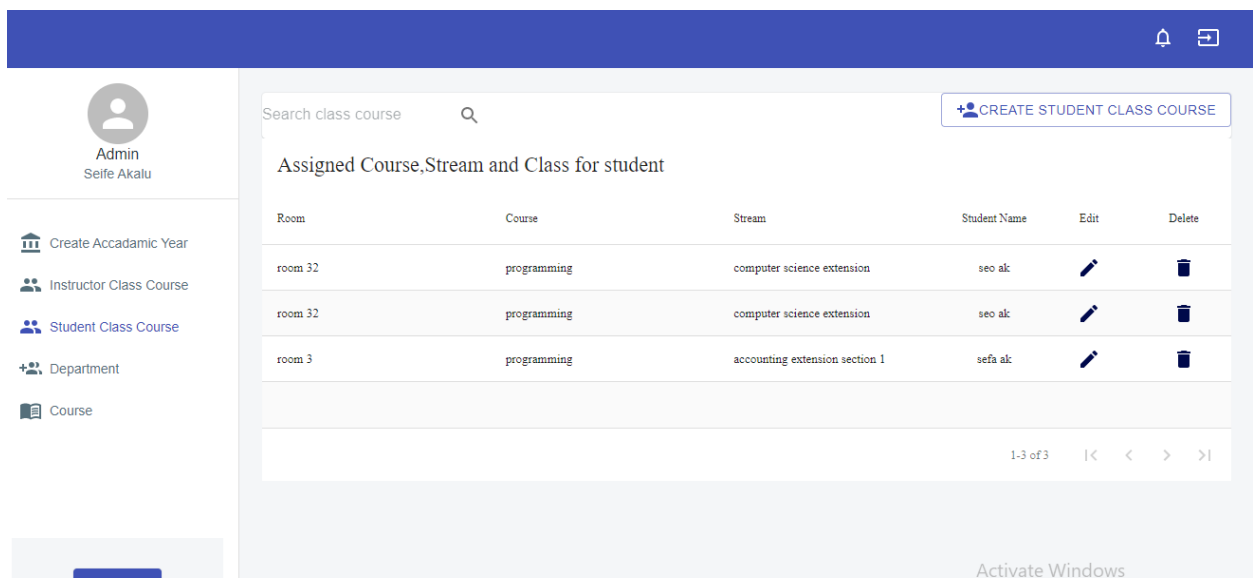


Figure 8 Student Class Course

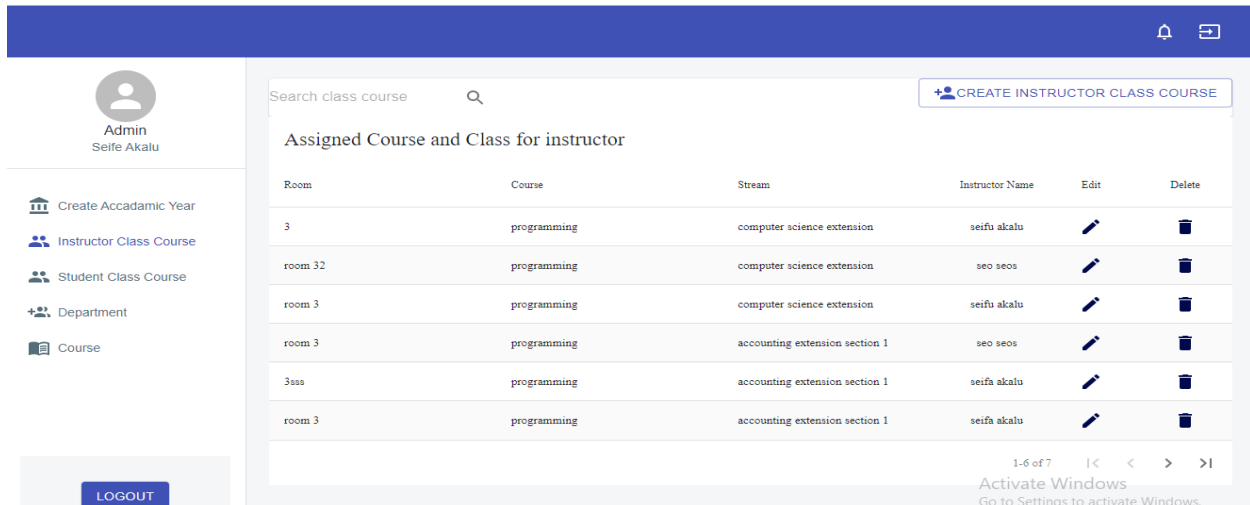


Figure 9 Instructor Class Course

4.2 Test Application Vulnerability

An API penetration test emulates an external attacker or malicious insider specifically targeting a custom set of API endpoints and attempting to undermine the security in order to impact the confidentiality, integrity, or availability of resources.

4.2.1 API Vulnerability Checklist

Table 1 Test Checklist

Vulnerability	Tool Used	Manual
SQL Injection	Vooki	Malicious Code Injection
Cross-site Scripting	Vooki	Malicious Code Injection
Brute Force	Vooki	--
Hash Password	Vooki	--
Encryption/Decryption	Postman	Database Value
Authentication/Authorization	--	Check Each Routs
Valid Cookie	Vooki	--
Reverse Engineering	--	Check the Front-end code
DOS Attack	Vooki	--
CSRF Token	--	Check on the Browser

4.2.2 Routs of the Application

This are routs of the application:

```
const department = require("../controllers/department.controllers");
const stream = require("../controllers/stream.controllers");
const student = require("../controllers/student.controllers");
const instructor = require("../controllers/instructor.controllers");
const year_of = require("../controllers/year_of.controllers");
const semester = require("../controllers/semister.controllers");
const classes = require("../controllers/class.controllers");
const instructor_class = require("../controllers/instructor_class_course.controllers");
const admin = require("../controllers/admin.controllers");
const course = require("../controllers/course.controllers");
const class_course = require("../controllers/class_course.controllers");
const student_class_course = require("../controllers/student_class_course.controllers");
```

4.2.3 Login Attempts and Hash Password

Using Vooki, login attempts was made to the developed Secure API, (Figure 10) shows request type and request Body to the API, and for successful login attempt it returns valid account information and Hash password as shown in Figure 11.

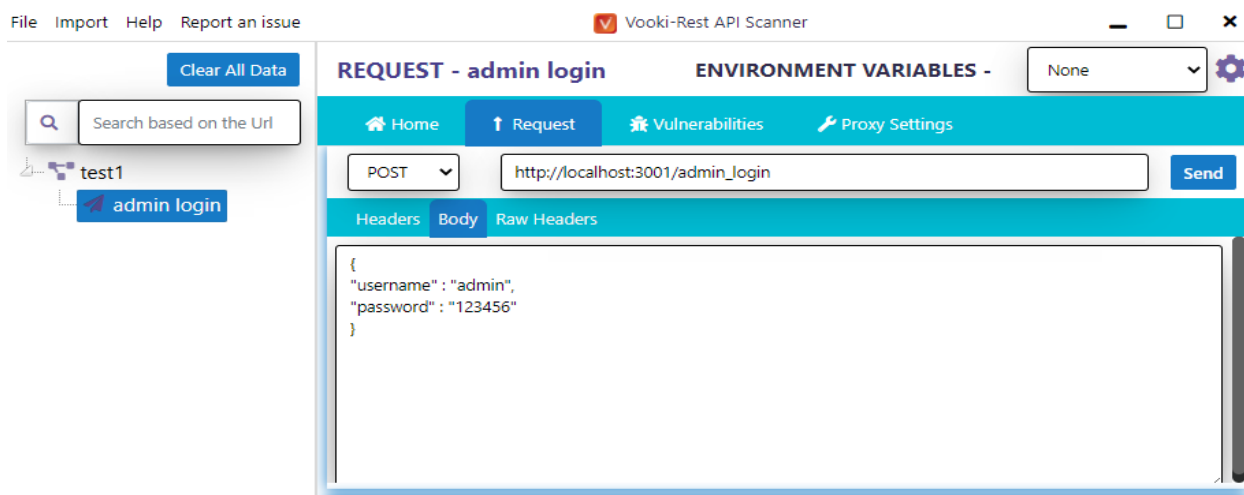


Figure 10 Login API Testing



Figure 11 Login Success result

4.2.4 Setting Valid Cookie After login attempts

After successful login attempt, the API Stores HTTP Only Cookie to the browser with expiration time. As shown in the image (Figure 12), This Cookie will only be transmitted by HTTP request and no other Front-end JavaScript code can access the Cookie.

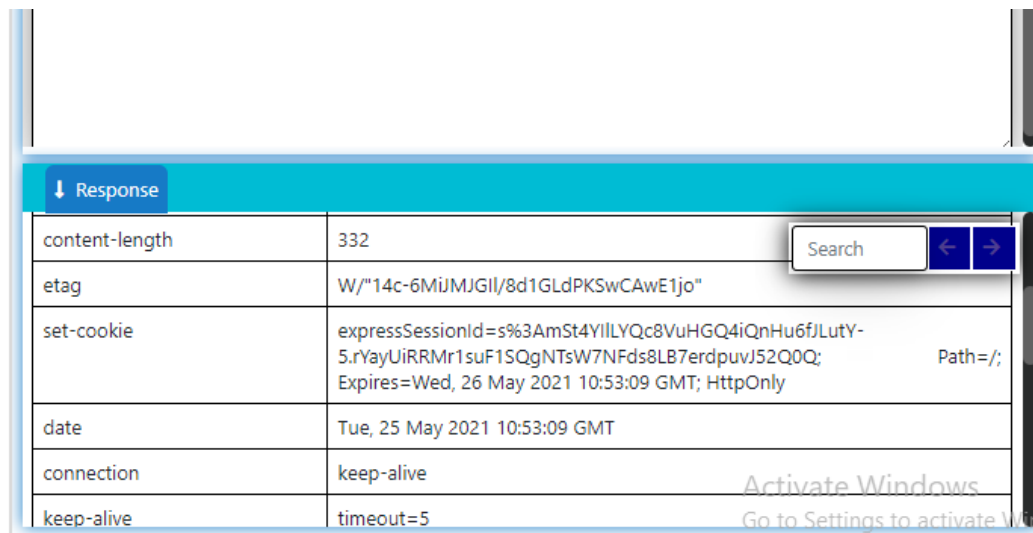
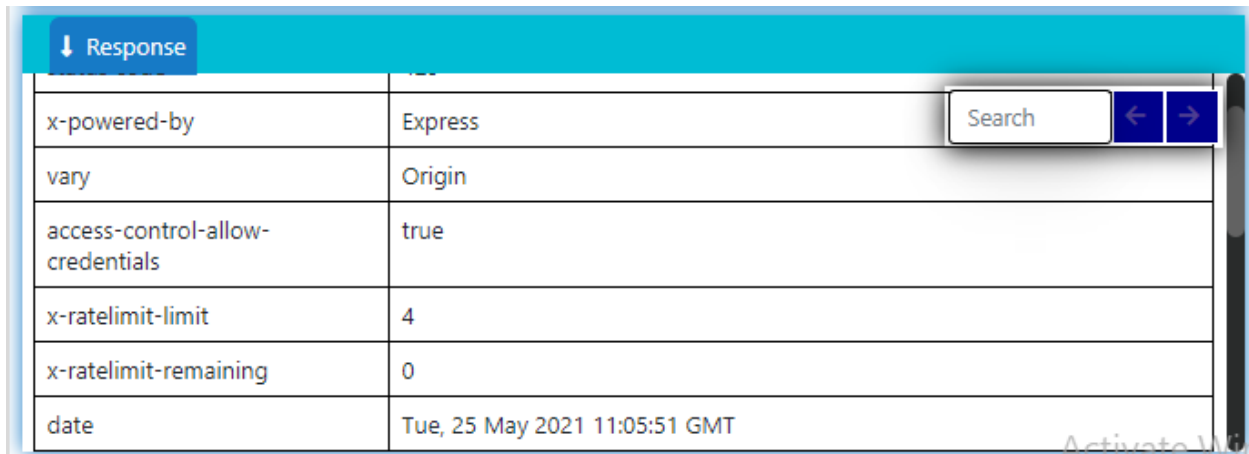


Figure 12 Setting Cookie


4.2.5 Rate Limiter for Brute-force attack

As shown in the image (Figure 13), the rate limiter will accept four requests within two minutes, “x-ratelimit-limit” field in the image shows the maximum request allowed within two minutes and “x-ratelimit-remaining” shows the current available request. After the remaining limit expires, the API displays error code (Figure 14) and now the application will not allow additional request.



↓ Response	
x-powered-by	Express
vary	Origin
access-control-allow-credentials	true
x-ratelimit-limit	4
x-ratelimit-remaining	0
date	Tue, 25 May 2021 11:05:51 GMT

Figure 13 Rate Limiter Limit



↓ Response	
etag	W/ 40-ngGLthm9Id7RQ8L3NYIKS1KCT4Y
connection	keep-alive
keep-alive	timeout=5

Response body	
<pre>{ "code": 429, "message": "Toomany login requests, try again later" }</pre>	

Figure 14 Error code After Limit Expires

4.2.6 Rate Limiter for DOS Attack

To protect the application from multiple requests, the new framework uses rate limiter to block any incoming requests. The API checks if user's request Per IP address expires and block any incoming requests from that IP Address. As shown on Figure 15, the application blocks if there is any incoming request with IP address and will display error message (Figure 16)

status code	429	Search	←	→
x-powered-by	Express			
x-ratelimit-limit	5			
x-ratelimit-remaining	0			
date	Tue, 25 May 2021 12:40:11 GMT			
x-ratelimit-reset	1621946487			

Figure 15 Rate Limiter DOS limit

Response body
<pre>{ "code": 429, "message": "Toomany requests, try again later" }</pre>

Figure 16 Response message if DOS Expires

4.2.7 AES Encryption

For showing purpose, this study tries to encrypt phone number during the instructor is registering and phone number stores to the database as shown on the below image.

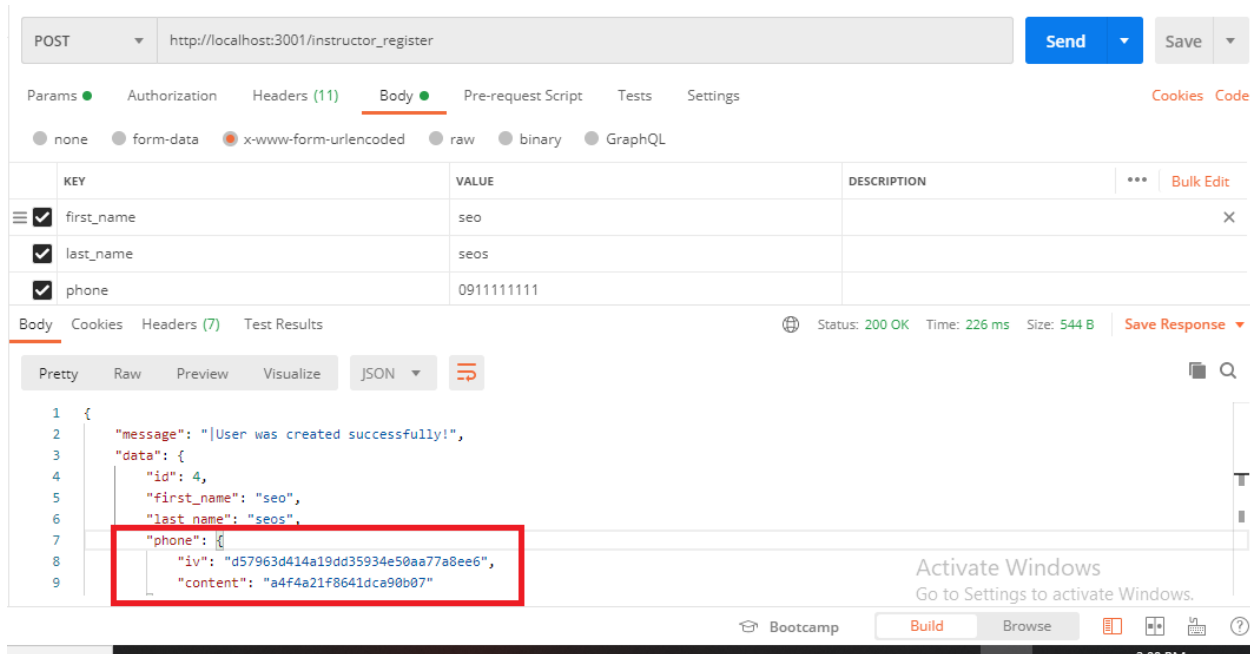


Figure 17 Encryption of Instructor Phone number

4.2.8 General Report From Vooki

The below image, (Figure 18), shows report generated from Vooki. And SQL Injection and Cross-site scripting was not found by this Tool.

SUMMARY OF FINDINGS (Scanned Node: admin login)

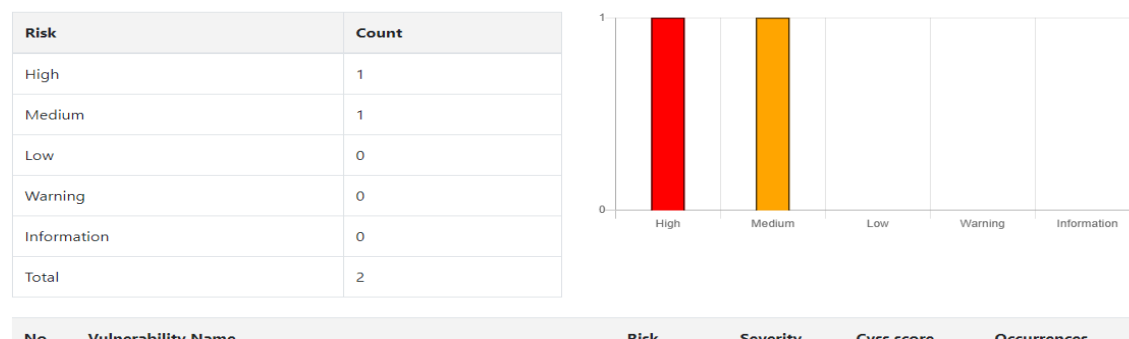


Figure 18 Vulnerability Test Report

No	Vulnerability Name	Risk	Severity	Cvss score	Occurrences
1	Insecure communication	High	High	8.1	1
2	Sensitive information disclosure in response headers	Medium	Medium	5.0	1

Figure 19 List of Vulnerability Found

4.2.9 Generated CSRF Token for every Page Request

For every page generated, new CSRF token will be generated and based on that CSRF Token, any Create, Read Update or Delete Request will be validated. The below image shows Cookie value of CSRF token

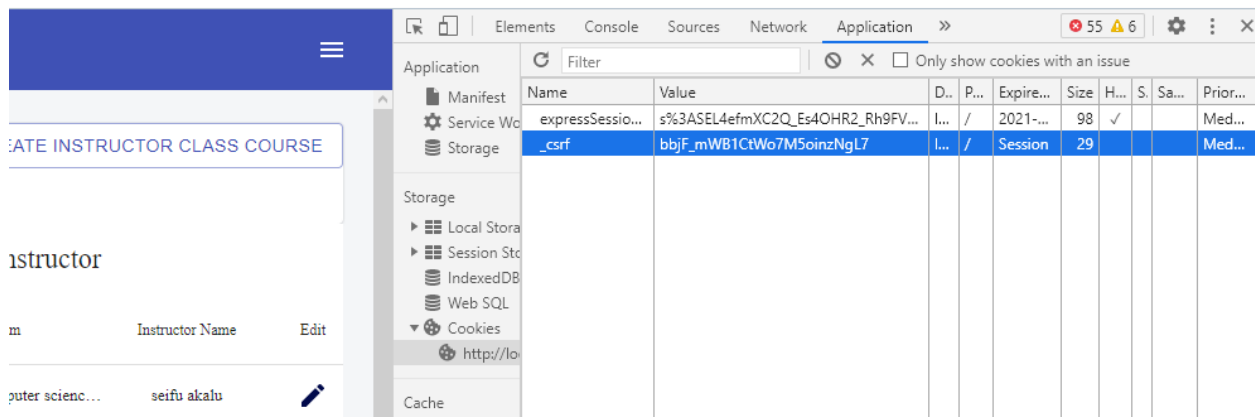


Figure 20 CSRF Token

4.2.10 SQL Injection With Malicious Code

If we remove, mapping of values in the array to placeholder like below SQL Injection is Possible and application will login successfully with the image shown (Figure 21, Figure 22)

```
"SELECT * FROM admin WHERE username = '"+admin.username+"' AND password = '"+admin.password+"'",
```

Sign in As Admin
Sign in with admin username and password

Username
' OR 1=1 -- -

Password
...

SIGN IN NOW

Figure 21 SQL injection Malicious Code

Search department

Admin
Seife Akalu

- Create Accademic Year
- Instructor Class Course
- Student Class Course
- Department
- Course

Customer Name	Description
business	business management
accounting	accounting dept
computer sciecne	works fine

1-3

Figure 22 Login Success

But if we map values in the array, the web application will display incorrect username or password as shown on the below image (figure 23)

```
`SELECT * FROM ${tableName} WHERE username = ? AND password = ?`,
  [admin.username, admin.password],
  function (error, results, fields) {
    if (results.length > 0) {
      console.log(results.data);
      result(null, { results, studentLogged: false, instructorLogged: false
, adminLogged: true, success: true } );
    }
  }
```

Incorrect username or password

Sign in As Admin

Sign in with admin username and password

Username

' OR 1=1 -- -

Password

.....

SIGN IN NOW

Figure 23 SQL Injection Error

4.3 Why Framework is needed for JavaScript

Most past studies have done a framework and did secure practice technique with PHP and other programming language also. However, most old programming languages are not efficient as new ones, According to a Journal (Soomro, June 2017), old programming language like PHP, is synchronous, synchronous application block other I/O operations until it receives response from remote server, so this part of the study tries to show performance gap between past secure frameworks and new developed frameworks. Because of this event-driven non-blocking I/O modeling of JavaScript, everything happens at once, making Node.js incredibly quick. On the other hand, PHP is synchronous using multi-threaded blocking I/O that runs parallel to each other. It needs to wait for a function to return before proceeding to the next line to execute. jsonplaceholder.typicode.com was used to test the asynchronous and synchronous behavior and speed of both programming language because it is free online fake API that accepts request and return response this will be helpful for performance testing of both programming language

Table 2 Test Cases

Test Script	Number of trades	Tool used
Sample API/ PHP script	2/60	Jmeter
Sample API/ javascript	2/60	Jmeter

The above table (Table 2) shows, performance of Login API of both JavaScript and Sample PHP code will be tested with 40 and 60 request trades respectively by using Jmeter tool.

4.3.1 PHP Code

For showing how PHP perform tasks synchronously sample request to jsonplaceholder.typicode.com was made to show performance gap between PHP and JavaScript. The below code shows how 100 request was made to API and its response,

```
<?php
$response_data='';
$json_data='';
$user_data='';
$api_url='';
$results =[];
    for($i=0; $i<100; $i++){
        $api_url = 'http://jsonplaceholder.typicode.com/comments';
        $json_data = file_get_contents($api_url);
        array_push($results, $results, $json_data);
    }
$END =microtime( true);
$execution_time =($END - $_SERVER{'REQUEST_TIME_FLOAT'});
echo "TIme: ". $execution_time ." result length ". count($results);

?>
```

4.3.2 JavaScript Code

For showing how JavaScript perform tasks asynchronously sample request to jsonplaceholder.typicode.com was made to show performance gap between PHP and JavaScript. The below code shows how 100 request was made to API and its response

```
router.get("/", function (req, res, next) {
  console.log("called");
  var todos = [];
  var count = 100;
  function init() {
    for (let i = 0; i < count; i++) {
      fetch(
        "http://jsonplaceholder.typicode.com/comments?cache=" +
        new Date().getTime()
      )
        .then((response) => response.json())
        .then((response) => {
          todos.push(response);
          credebttials();
        })
        .catch(function () {
          console.log("error");
        });
    }
  }
  function credebttials() {
    if (todos.length < count) return;
    console.log("All done");
  }
  init();
});
```

4.3.3 Setting Request

Before requesting using Jmeter, there must be things set like URL, Port number, Number of trades must be set. This part discusses these Settings.

4.3.3.1 Set Request URL and Port Number

For successful request using Jmeter, the API URL and Port number must be set so that request can be redirected to that URL and Port Number the below image shows request URL and Port Number Setup

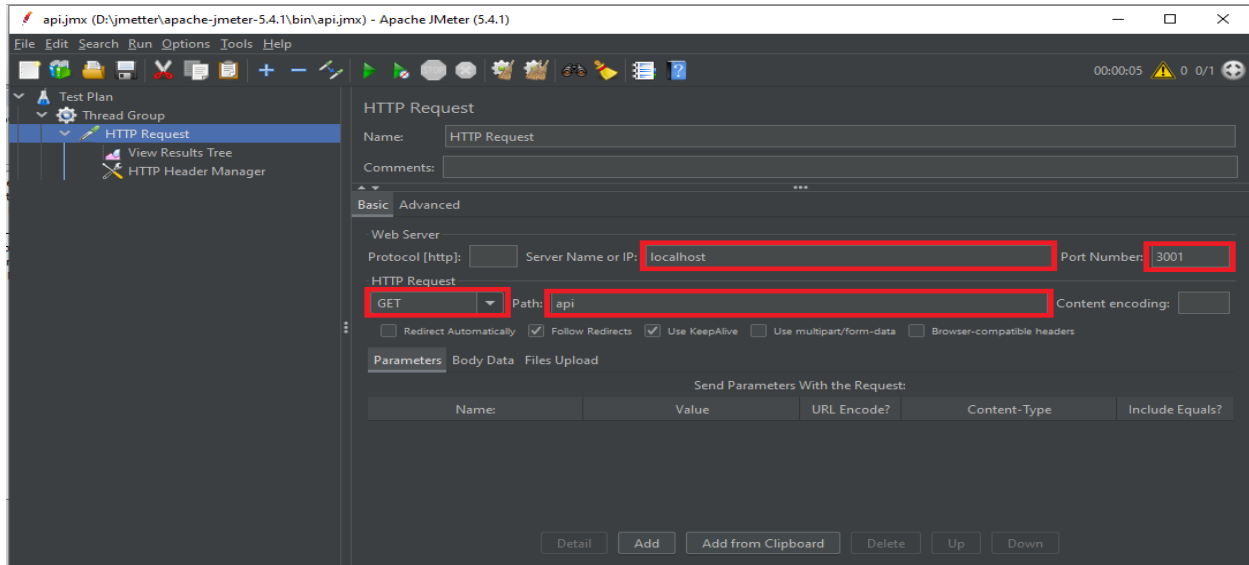


Figure 24 Request URL

4.3.3.2 Set Number of Trades

For testing the API, the Load it can handle, number of trades must be set so it will request for concurrent request to the API and will show the performance. The below image shows how number of trades can be set using Jmeter.

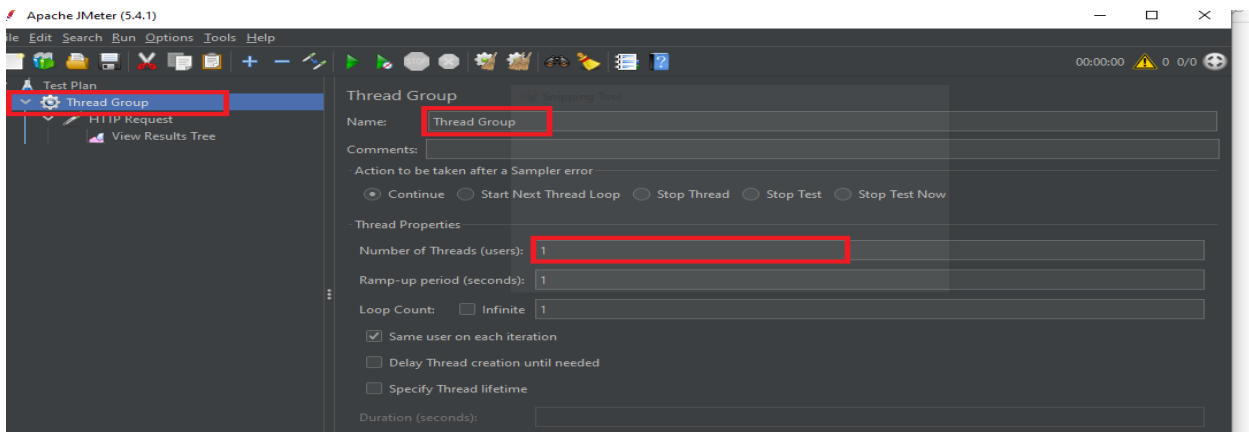


Figure 25 Number of trades

4.3.3.3 Set Result Tree

Result tree helps to show each request and their performance measure. For example, if we have 40 trades we can get each separate requests performance the below image show result tree of requests

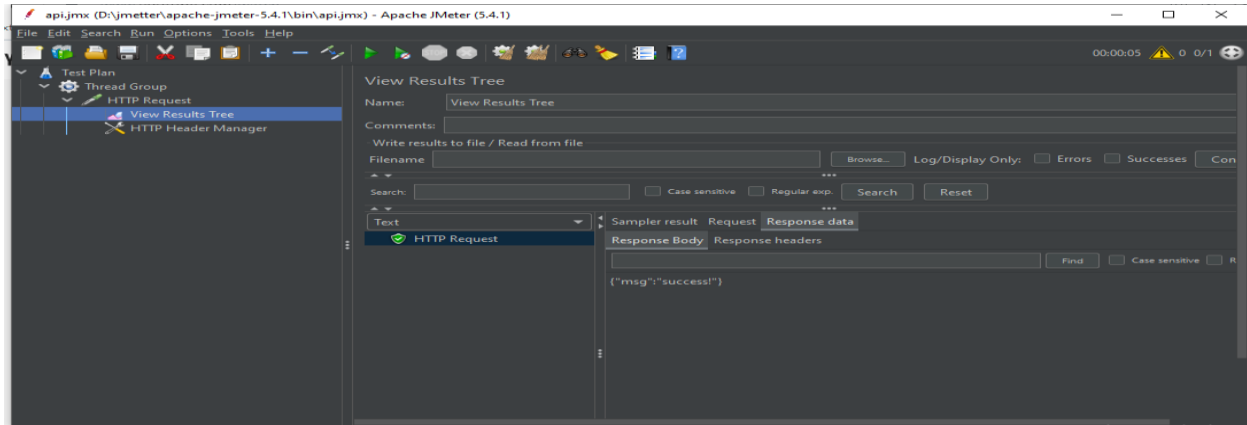


Figure 26 Result tree

4.3.4 Generated Performance report

This part shows performance measure of the new sample code by comparing with PHP programming and show statistics of the performance

4.3.4.1 PHP Script Performance with 40 Trades

By using Jmeter with 1 trade report was generated for PHP Script to fetch from sample API and the below image (Figure 27, Figure 28) shows result of the test.

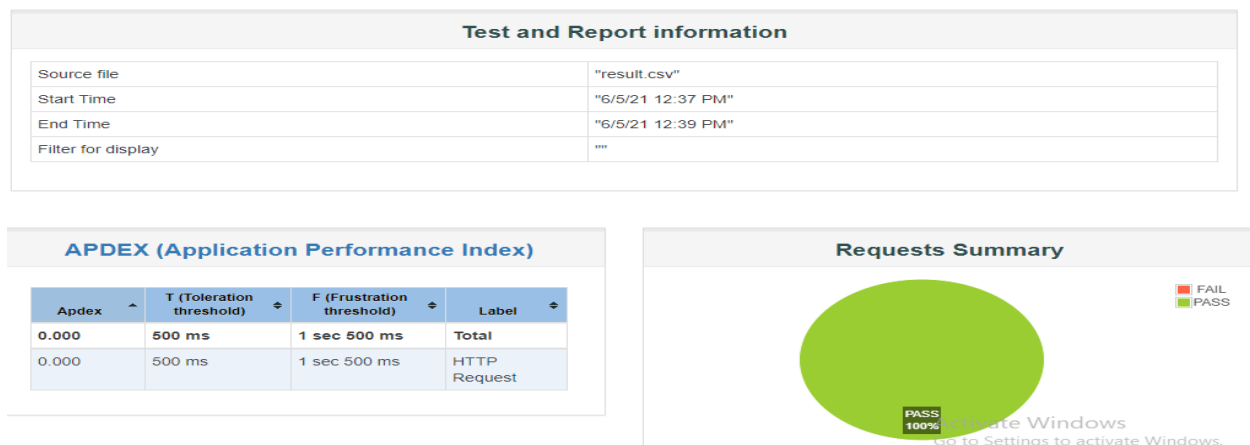


Figure 27 PHP Report from HTML

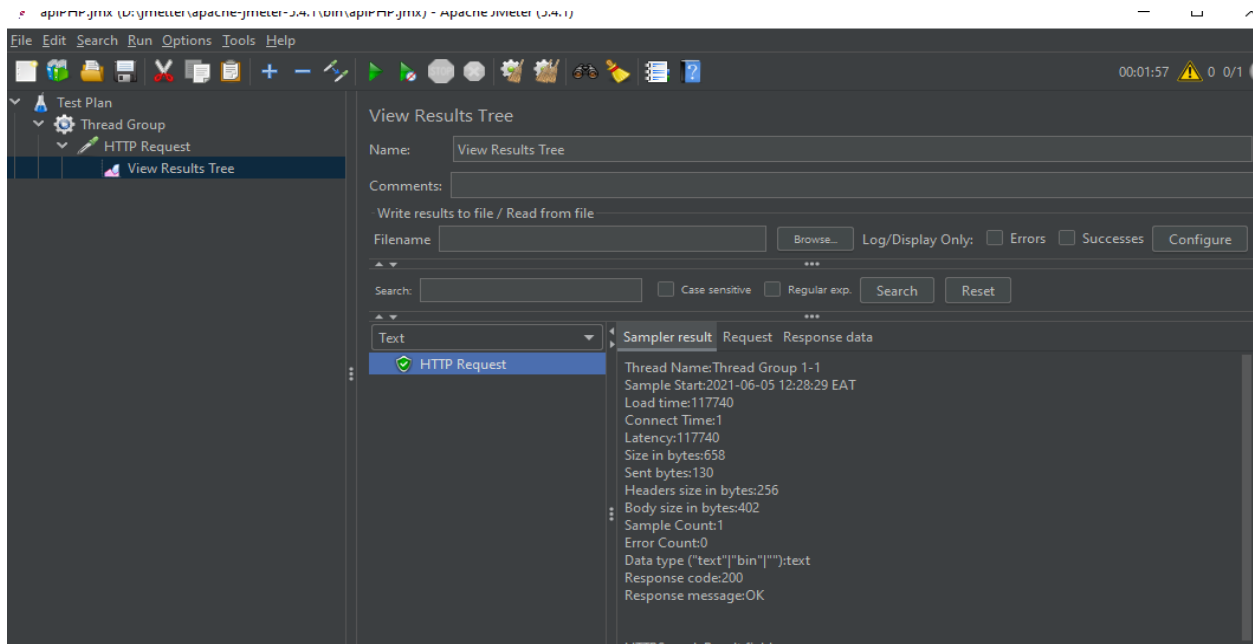


Figure 28 Load time of PHP Script

4.3.4.2 JavaScript Performance with 40 Trades

By using Jmeter with 1 trade report was generated for JavaScript Script for fetching from API and the below image (Figure 27, Figure 28) shows result of the test.

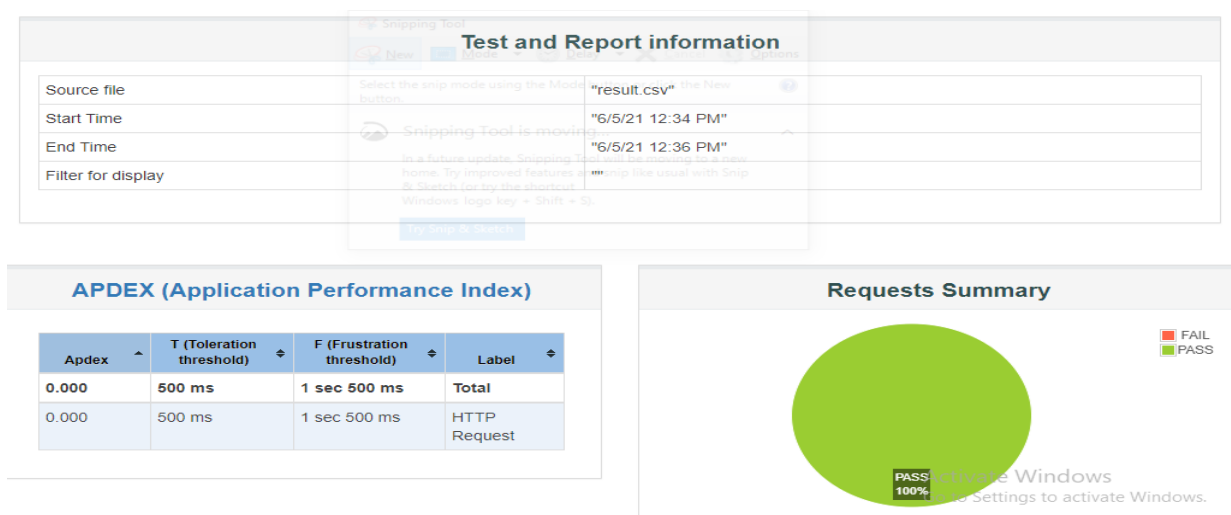


Figure 29 JavaScript Report from HTML

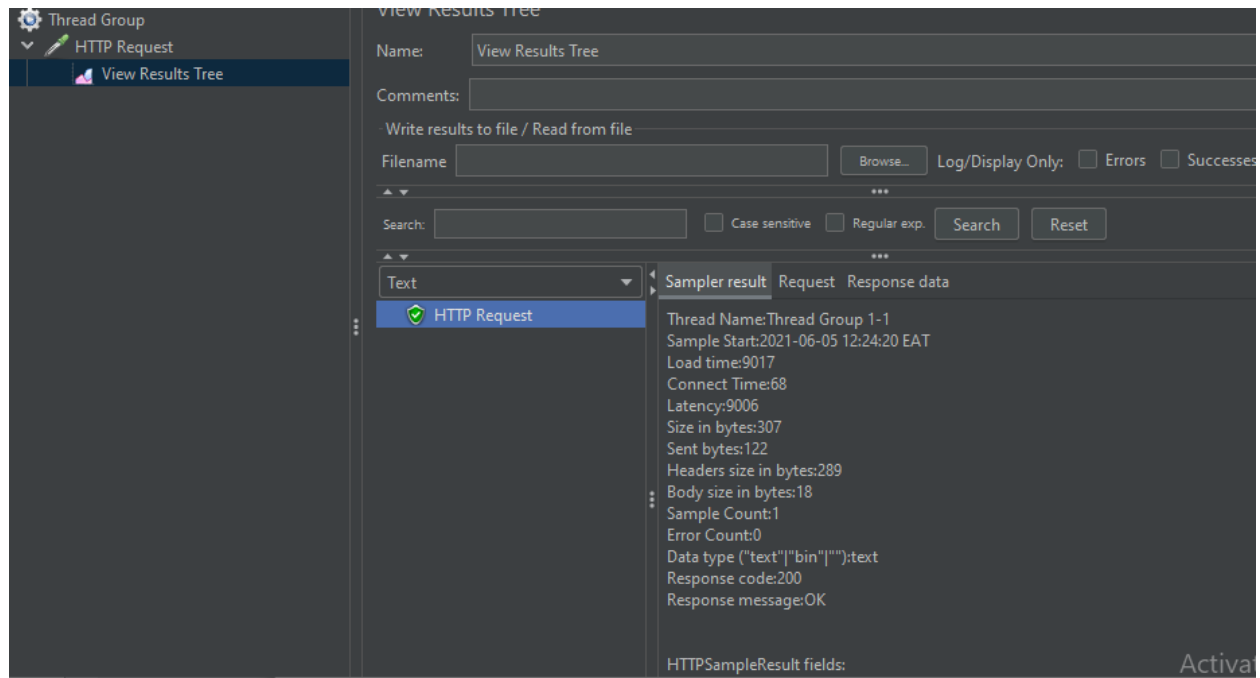


Figure 30 JavaScript Load time

4.3.4.3 Summary of Result

After testing JavaScript and PHP with 100 requests to online free API jsonplaceholder.typicode.com, result shows load time of 117740 for PHP and for JavaScript 9071 which means JavaScript is faster when requesting because node is Asynchronous and PHP is Synchronous.

4.3.5 Performance Comparison of SQL Injection of JavaScript and PHP

This part of the paper shows how JavaScript performs better than PHP with concurrent multiple requests. Both have the same techniques which registers sample data to database. For JavaScript, the algorithm uses SQL injection techniques provided in the framework and for PHP, it uses techniques suggested in past studies and shows how JavaScript performs better especially for application that are highly dependent on client-server network.

4.3.5.1 PHP Code

For filtering any SQL Injection query the below PHP code Uses MySql escape string this will be helpful to escape any malicious code from user input. After user data successfully filtered the code inserts 100 data to MySql table.

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "script";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
$name=$conn -> real_escape_string("seife");
$age=$conn -> real_escape_string(29);
for($i=0; $i<100; $i++){
    $sql = "INSERT INTO register (name, age)
    VALUES ('".$name."', '".$age."')";

    if ($conn->query($sql) === TRUE) {
        echo "New record created successfully";
    } else {
        echo "Error: " . $sql . "<br>" . $conn->error;
    }
}

$conn->close();
?>
```

4.3.5.2 JavaScript Code

For filtering any SQL Injection query the below JavaScript code Uses MySql escape string. Attribute names inside the object become the placeholders in the SQL query. This means any characters that include SQL syntax will be considered as part of the overall SQL query so application is protected from SQL Injection.

```

Customer.create = (customer, result) => {

  for(var i=0; i<100; i++){
    const sql = `INSERT INTO ${tableName} SET ?`;
    db.query(sql, customer, (err, res) => {
      if (err) {
        result(err, null);
        return;
      }

    });

  });

  result(null, { id: 1, ...customer });
};

```

4.3.5.3 Setting HTTP Request

For requesting the Code to perform the task this study uses JMeter and did request for both PHP and JavaScript code and did comparison on performance. Figure 31 shows request setting for PHP and Figure 32 shows request URL setting for JavaScript

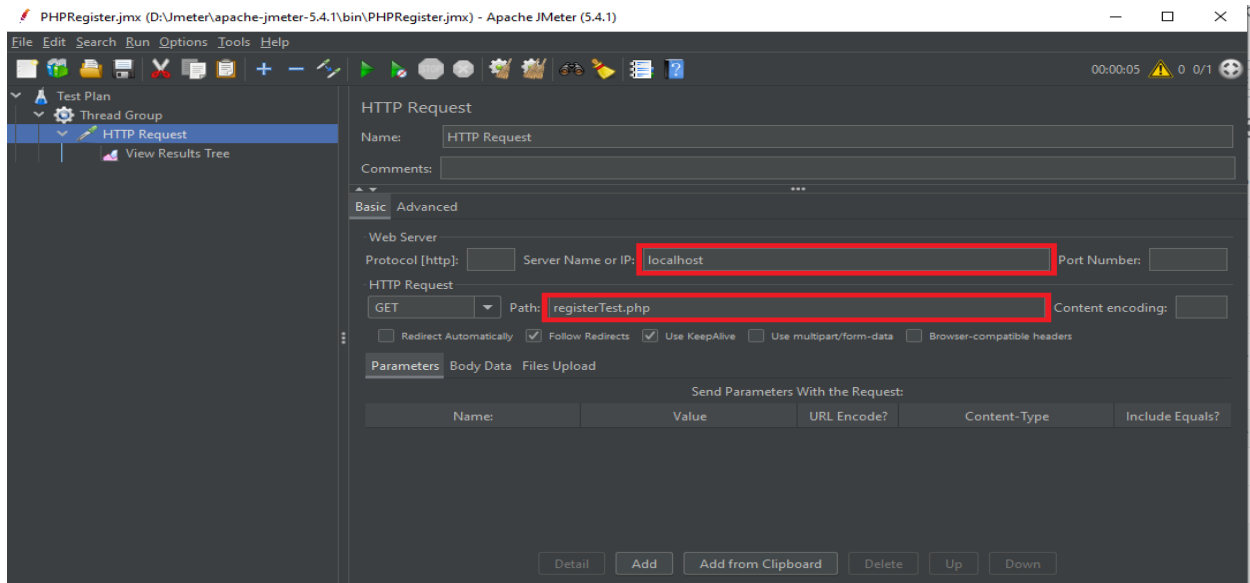


Figure 31 PHP HTTP Request

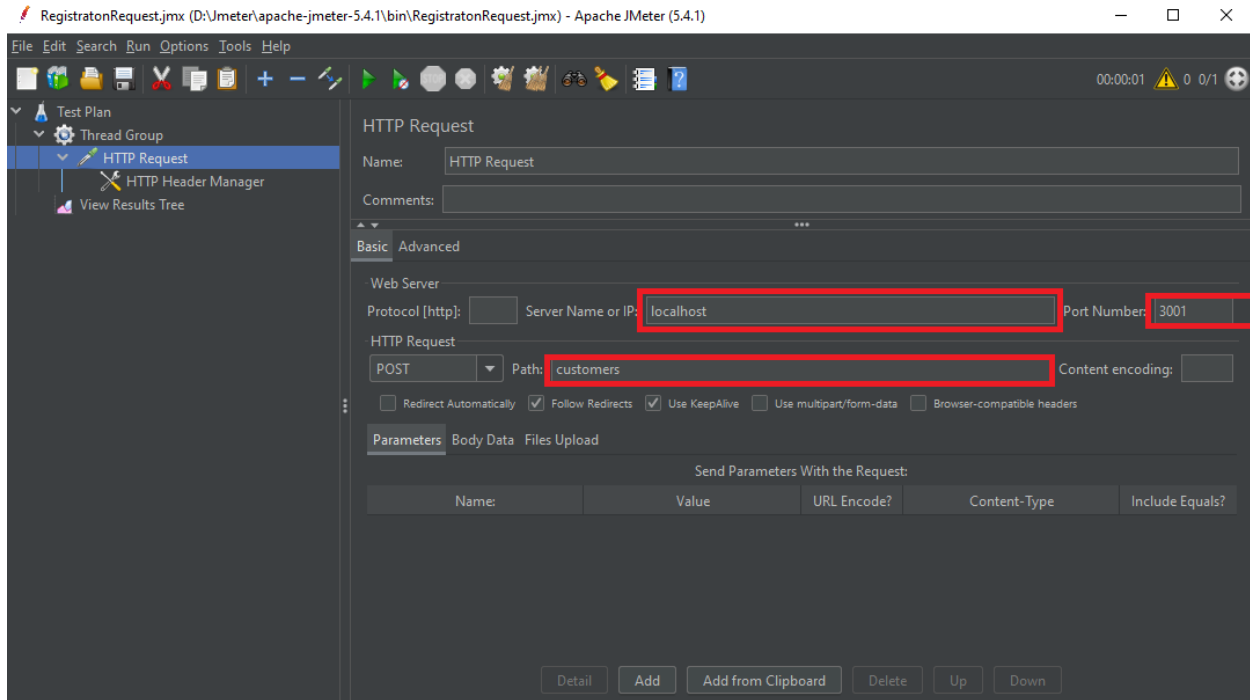


Figure 32 JavaScript HTTP Request

4.3.5.4 Setting Traded Group

The traded group section will help to sate number of concurrent request needed for the request. And for testing Purpose this study uses only 1 trade request for both PHP and JavaScript.

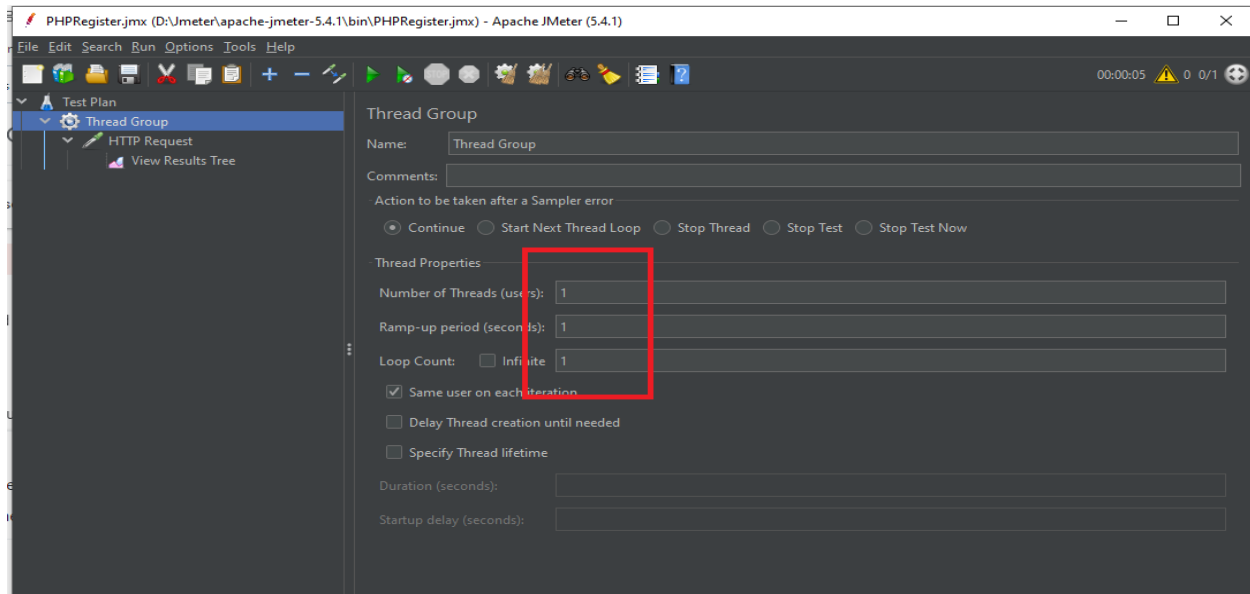


Figure 33 PHP Traded Group

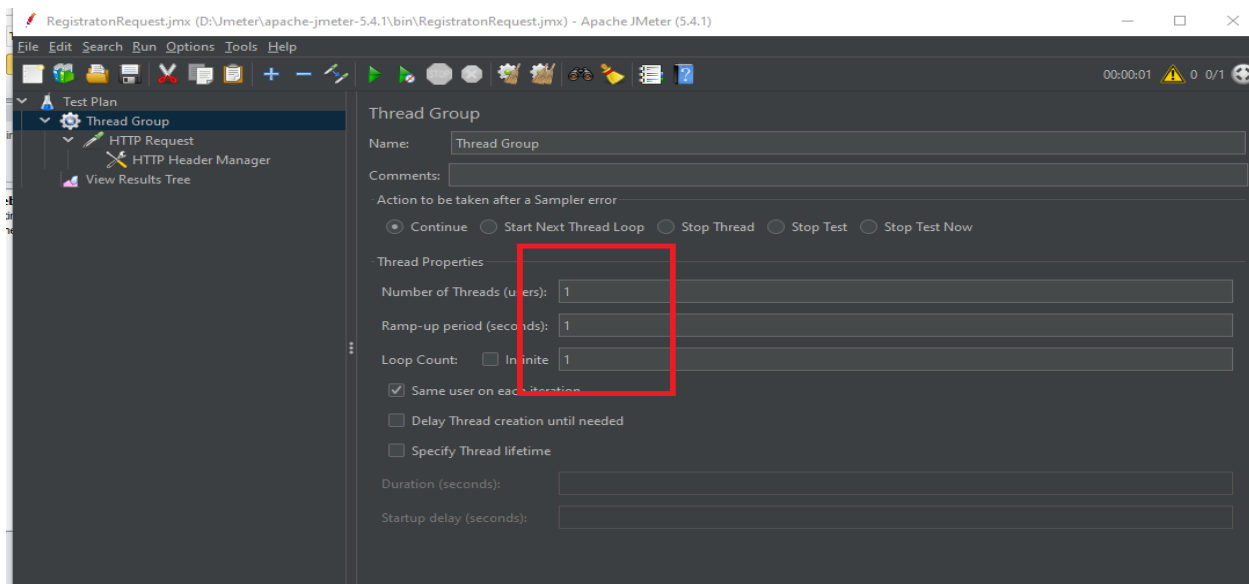


Figure 34 JavaScript Traded Group

4.3.5.5 Showing Performance Result

This study Compares SQL injection security technique used in the new framework and from past studies and show the performance by using JMeter. As you can see in the below image, JavaScript performs better and JMeter Result shows both Response time and Latency of JavaScript Code is so much better.

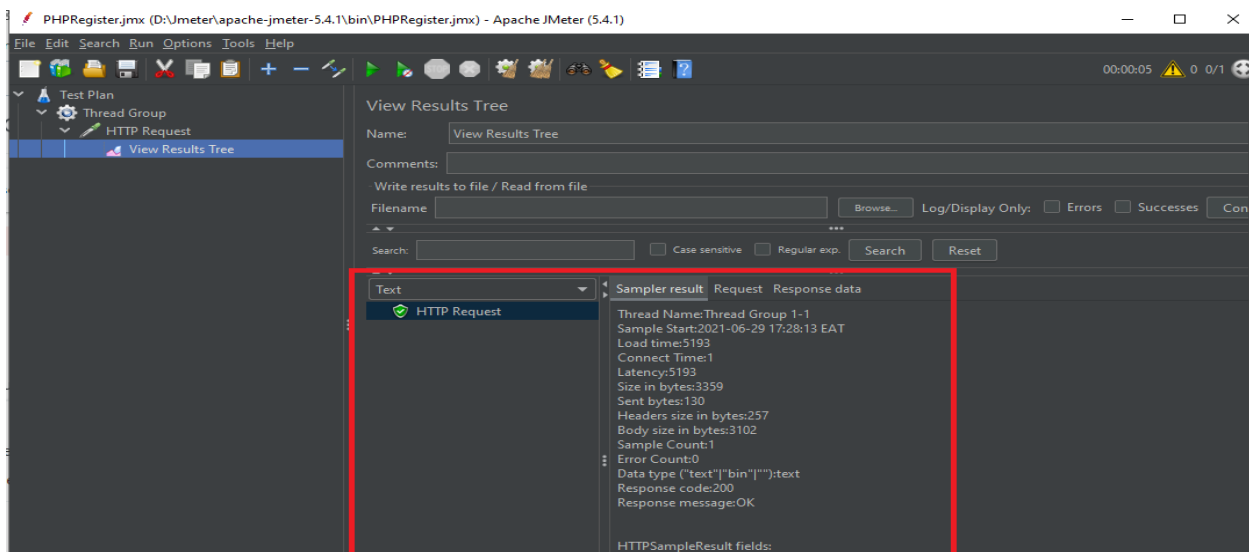


Figure 35 PHP Response time

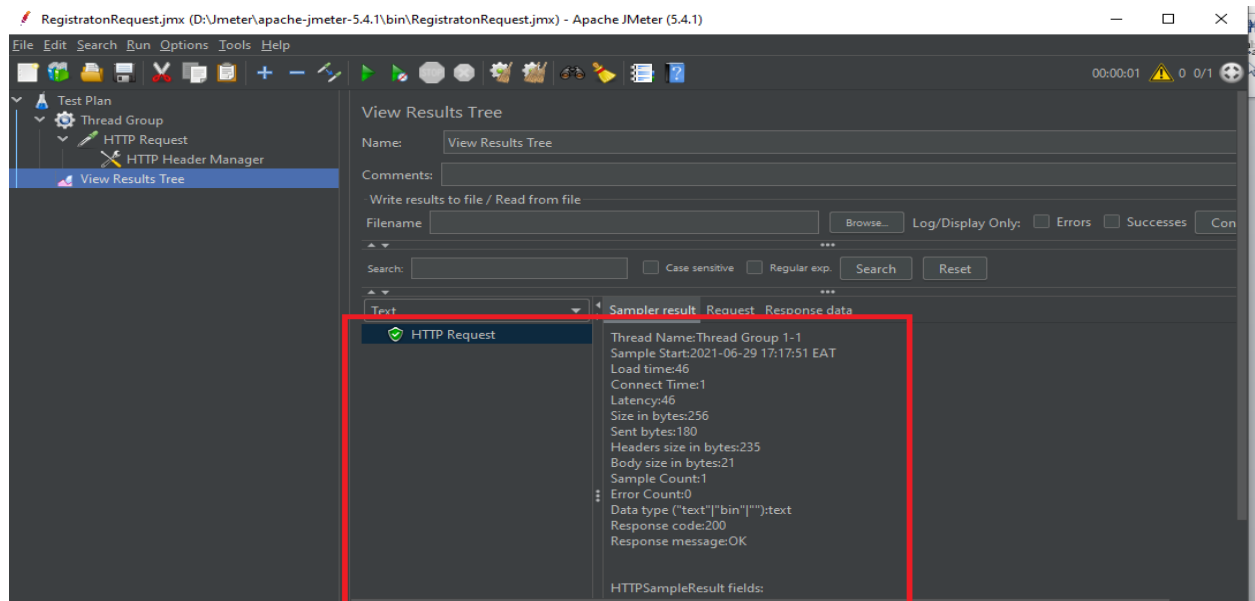


Figure 36 JavaScript Response time

CHAPTER FIVE

CONCLUSTION AND RECOMMENDATION

5.1 Introduction

This Chapter of the research shows conclusions and recommendation based on the result shown on chapter four. This chapter specifically discuss why JavaScript needs security framework and is the discuss JavaScript is the future of Web Application development.

5.2 Conclusion

To design web application security that includes proper way of authorization and authentication that manifests best coding practice, especially, for developers and software engineers, this research has studied various attack types and done secure framework. On the way, this study has found JavaScript to be one of the most powerful languages on the planet because of its performance and omnipresence. Because of this, JavaScript needs security framework and for this reason, security framework was developed for JavaScript and most common web application vulnerabilities had been tested by using manual and tool based techniques. Additionally, by testing the performance of old secure framework and the new one, this research has found JavaScript better in terms of performance because it has lower latency during network request so this study has concluded JavaScript to be better performs in terms of handling request to the network. As seen on chapter four, by using JavaScript security framework, most common web application vulnerability can be protected by using best Coding practice.

5.3 Recommendation

This research shows how common web application vulnerability can be protected by using JavaScript framework. However, further research should be done regarding all web application vulnerability and prepare open source JavaScript security framework.

Reference

- Al-Ibrahim, M. (2014). *The Reality of Applying Security in Web Applications in Academia* (Vol. 5). (IJACSA) International Journal of Advanced Computer Science and Applications.
- Anis, A. (2018). *Securing web applications with secure coding practices and integrity verification*. Kingston, Ontario, Canada: Queen's University.
- Apache. (2021). *JMeter*. Retrieved from Apache JMeter: <https://jmeter.apache.org>
- Atiqur Rahman, M. M. (2015). Security Assessment of PHP Web Applications from SQL Injection Attacks., *volume 6*.
- Clarinsson, R. (2005). *The PHP programmer's guide to secure*. Richard Clarinsson.
- Facebook. (2021). *React*. Retrieved from Official Docs: <https://reactjs.org>
- Forsman, T. (2014). *Security in Web Applications and the Implementation of a Ticket Handling System*. SE-901 87 UME°A: Ume°a University.
- Giannini, N. J. (2015). *Vulnerable Web Application Framework amework*. Island: University of Rhode Island.
- Jingling Zhao, R. G. (2015). *A New Framework of Security Vulnerabilities Detection in PHP Web Application*. Beijing: University of Posts & Telecommunications.
- Jun Zhu *, J. X. (2013). *Supporting secure programming in web applications through interactive static analysis', Department of Software and Information Systems*. Charlotte, USA: University of North Carolina at Charlotte.
- Nguyen, D. T. (2015). *Web Security: Security Methodology for Integrated Website using RESTful Web Services*. University of Tampere.
- OWASP. (2020). *The OWASP® Foundation* . Retrieved from OWASP OWASP Top Ten: <https://owasp.org/www-project-top-ten/>
- Pelizzi, R. (2016). *Securing Web Applications*. Stony Brook University.
- Savio Antony Sebastian, S. M. (2016 IEEE). A Study & Review on Code Obfuscation. *World Conference on Futuristic Trends in Research and Innovation for Social Welfare (WCFTR'16)*.
- Stefan, D. (2015). *PRINCIPLED AND PRACTICAL WEB APPLICATION SECURITY*. STANFORD UNIVERSITY.
- VegaBird. (2021). *VOOKI*. Retrieved from VOOKI - Free Vulnerability Scanner (DAST Tool): <https://www.vegabird.com/vooki/>
- Vincent Appiah, I. K.-B. (2017). Investigating Websites and Web Application Vulnerabilities: Webmaster's Perspective. *International Journal of Applied Information Systems (IJ AIS) – ISSN : 2249-0868 , 12*.

Walley, K. (2019, November 12). *Frameworks for Web Development 2020*. (dev.to) Retrieved from DEV:
<https://dev.to/websitedesignnj/top-6-most-used-php-frameworks-for-web-development-2020-46o5>

Xiaowei Li, Y. X. (2014). A Survey on Web Application Security.