

A Study on Payment Gateway Integration, Security Analysis, and Vulnerability Mitigation of Enterprise E-Commerce Payment Systems

submitted by

Dwip Shekhar Mondal

Class Roll No: 002210503037

Exam Roll: MCA244036

Master of Computer Application

under the supervision of

Prof. (Dr.) Subhadip Basu

in the

Department of Computer Science and Engineering

Faculty of Engineering and Technology

Jadavpur University

2024

Faculty of Engineering and Technology Jadavpur University

CERTIFICATE OF RECOMMENDATION

To Whom It May Concern

I hereby recommend that the project report entitled "A Study on Payment Gateway Integration, Security Analysis, and Vulnerability Mitigation of Enterprise E-Commerce Payment Systems" has been carried out by Dwip Shekhar Mondal (Reg. No.: 163629 of 2022-2023, Roll No.: 002210503037, Exam Roll No.: MCA244036) under my guidance and supervision and be accepted in partial fulfillment of the requirement for the Degree of Master of COMPUTER APPLICATION in DEPARTMENT of COMPUTER SCIENCE and ENGINEERING, JADAVPUR UNIVERSITY during the academic year 2023-2024.

Prof. (Dr.) Subhadip Basu
Project Supervisor
Dept. of Computer Science and Engineering
Jadavpur University

Prof. (Dr.) Nandini Mukhopadhyay
Head of the Department
Dept. of Computer Science and Engineering
Jadavpur University

Prof. Dipak Laha
Dean, Faculty Council
of Engineering and Technology
Jadavpur University

Faculty of Engineering and Technology
Jadavpur University

CERTIFICATE OF APPROVAL

This is to certify that that the project entitled "**A Study on Payment Gateway Integration, Security Analysis, and Vulnerability Mitigation of Enterprise E-Commerce Payment Systems**" is a bonafide record of work carried out by DWIP SHEKHAR MONDAL in partial fulfilment of the requirement for the award of the Degree of Master of Computer Application in the Department of Computer Science and Engineering, Jadavpur University. It is understood that by this approval, the undersigned does not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approves the project only for the purpose it has been submitted.

Signature of Examiner 1

Date:

Signature of Examiner 2

Date:

Faculty of Engineering and Technology
Jadavpur University

**DECLARATION OF COMPLIANCE WITH ACADEMIC
ETHICS**

I hereby declare that this project entitled "**A Study on Payment Gateway Integration, Security Analysis, and Vulnerability Mitigation of Enterprise E-Commerce Payment Systems**" is an outcome of my genuine and independent work, as part of my Master of Computer Application program.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, all materials and results that are not original to this work have been fully cited and referenced.

Dwip Shekhar Mondal

Registration No: 163629 of 2022-2023

Examination Roll No: MCA244036

Thesis title: A Study on Payment Gateway Integration, Security Analysis, and Vulnerability Mitigation of Enterprise E-Commerce Payment Systems

Signature with Date

ACKNOWLEDGEMENT

With my most sincere and gratitude, I would like to thank **Dr. Subhadip Basu**, Department of Computer Science and Engineering, Jadavpur University for his admirable guidance, care, patience and for providing me with an excellent atmosphere for doing this project. Our numerous discussions and his many constructive comments have greatly improved this work.

I want to express my sincere gratitude to the team at **Infomaticae Technology Pvt Ltd** for their guidance and cooperation and for providing me with a platform to work on research and development initiatives for improving the effectiveness and security of payment systems. The project's successful completion was made possible in large part by their guidance and collaboration.

I am highly grateful to my mentor **Mr. Apurba Pramanik** at Infomaticae Technology Pvt Ltd for his guidance from the start and throughout the completion of the project. I Would also like to thank my Friends who were always present to motivate me. Most importantly none of this would have been possible without the love and support of my family. I extend my thanks to my parents, especially to my mother whose forbearance and wholehearted support helped this endeavor succeed.

This project would not have been completed without the inspiration and support of a number of wonderful individuals, my thanks and appreciation to all of them for being part of this journey and making this project possible.

Dwip Shekhar Mondal

Class Roll No: 002210503037

Examination Roll No: MCA244036

Department of Computer Science and Engineering

Jadavpur University

Contents

Table of Contents	i
List of Figures	iv
1 Introduction	1
1.1 Background	1
1.1.1 E-commerce Payment Processing	2
1.1.2 Security Risks of E-commerce Payment Systems	2
1.1.2.1 Malware	3
1.1.2.2 Man-in-the-middle attacks	3
1.1.2.3 Distributed Denial of Service (DDoS) Attacks	4
1.1.2.4 Remote Code Execution	4
1.1.2.5 Cross site Scripting (XSS) attacks	4
1.1.2.6 SQL Injection	4
1.2 Motivation	5
1.3 Scope of Current Work	5
1.4 Survey of Related Works	6
1.5 Organization of the Report	7
2 Practical Concepts and Approaches to Development	9
2.1 Web Applications	9
2.1.1 Web Application Architectures	9
2.1.1.1 Model View Controller Architecture (MVC)	10
2.1.2 Web Development Technologies	10
2.2 Web-Application Security Testing	11
2.2.1 Types of Security Testing	12
2.3 Security Testing and Assessment Tools	13
2.3.1 Burp Suite	13
2.3.2 Nmap	13
2.3.3 Wireshark	13
2.3.4 Nessus	14
2.3.5 Metasploit	14
2.4 Role of Cryptography in Web Security	14
2.4.1 Symmetric Key Cryptography System	15
2.4.2 Asymmetric Key Cryptography System	15
2.4.2.1 One Way Functions and Trapdoors	16

2.4.3	Digital Signature and Hashing	17
2.4.3.1	Modern Hash Function Standards	18
3	System Development and Payment Gateway Integration	19
3.1	Development Process Overview	19
3.1.1	Technologies and Tools Used in Development	19
3.2	Functional Requirements	20
3.3	Payment Gateway Integration	21
4	Security Assessment and Results	24
4.1	Security Assessment of Payment Service	24
4.1.1	Security Assessment Approach	24
4.2	Security Assessment Results	25
4.2.1	Manipulation of Payment Using Parameter Tampering . . .	25
4.2.1.1	Risk Description	25
4.2.2	Exposure of critical Information Using Insecure Direct Object Reference (IDOR)	27
4.2.2.1	Risk Description	27
4.2.3	HTML Injection Attack	28
4.2.3.1	Risk Description	28
4.2.4	Insecure Cookie Attribute Configuration	29
4.2.4.1	Risk Description	29
4.2.5	CORS Misconfiguration	30
4.2.5.1	Risk Description	32
5	Vulnerability Mitigation Procedures	33
5.1	Vulnerability Mitigation Process Overview	33
5.2	Mitigation of Discovered Vulnerabilities	34
5.2.1	Manipulation of Payment Using Parameter Tampering . . .	34
5.2.1.1	Possible Mitigation Measures	34
5.2.1.2	Mitigation Measures Taken	35
5.2.2	Exposure of critical Information Using Insecure Direct Object Reference (IDOR)	37
5.2.2.1	Possible Mitigation Measures	37
5.2.2.2	Mitigation Measures Taken	38
5.2.3	HTML Injection Attack	38
5.2.3.1	Possible Mitigation Measures	39
5.2.3.2	Mitigation Measures Taken	39
5.2.4	Insecure Cookie Attribute Configuration	39
5.2.4.1	Possible Mitigation Measures	40
5.2.4.2	Mitigation Measures Taken	40
5.2.5	CORS Misconfiguration	40
5.2.5.1	Possible Mitigation Measures	40
5.2.5.2	Mitigation Measures Taken	40

6	Conclusions	42
6.1	Limitations of Current Work	43
6.2	Scope for Improvements	43
A	Appendix-I	44
A.1	Screens of the Payment Service	44
A.2	Sample Code Snippets	47
	Bibliography	52

List of Figures

1.1	Example of an E-commerce Payment Process	2
1.2	Man in the middle attack	3
2.1	Model-View-Controller (MVC) architectural pattern [Image courtesy: Wikipedia, "Model-view-controller"]	10
2.2	E-commerce transaction through MVC process [Image courtesy: H. Nahari and R. L. Krutz, Web commerce security: design and development.]	11
2.3	Symmetric key cryptographic system	15
2.4	Asymmetric key cryptographic system	16
2.5	One way function	17
2.6	Trapdoor One way function	17
2.7	Process of digitally signing a message	18
3.1	Diagram of payment service	21
3.2	Process of online payment through payment service	22
3.3	Main payment screen	23
4.1	Parameter tampering attack	26
4.2	Insecure direct object reference attack example	27
4.3	IDOR attack process of this case	28
4.4	A Code injection attack escalating to an XSS attack [Image courtesy: Cross-site scripting (XSS) by Snyk Learn, url: https://learn.snyk.io/]	30
4.5	Cross-origin requests controlled by CORS configuration	31
4.6	Exmaple of a CORS misconfiguration exploitation [Image courtesy: D. Danelon, The complete guide to cors (in)security]	32
5.1	Request integrity check between payment service to payment gateway	36
5.2	Integrity check of response from payment gateway to payment service	36
5.3	Example of CORS configuration preventing external origin attacks [Image courtesy: D. Danelon, The complete guide to cors (in)security]	41
A.1	Initial payment screen on request to payment-link	44
A.2	User redirected to next screen on clicking pay now	45
A.3	Payment gateway for online payment	45
A.4	Fail screen on a failed transaction	46
A.5	Success screen on a successful transaction	46

A.6	Code snippet of payment gateway integration	47
A.7	Code snippet of integrity check of response from payment gateway (Client side)	48
A.8	Code snippet of getting order data using orderId and access token .	49
A.9	Code snippet of success screen along with response validity check .	50
A.10	Code snippet of failure screen along with response validity check . .	51

CHAPTER 1

Introduction

The rise of e-commerce and digital transactions in recent years has led to a great need for secure and efficient payment systems for enterprise e-commerce platforms. This study focuses on secure payment gateway integration, best practices and examines the security risks associated with enterprise e-commerce payment systems and practices to mitigate security vulnerabilities.

1.1 Background

The growth of online transactions is increasing at an exponential rate. In recent years, e-commerce has become a vital component of international retail. As global internet access and adoption rapidly increase, with over five billion internet users worldwide, the number of people making purchases online is ever-increasing. In 2024, retail e-commerce sales are estimated to exceed 6.3 trillion U.S. dollars worldwide, and this figure is expected to reach new heights in the coming years. [1]

As Online transactions have become the new norm, the security risks associated with it have increased at a matching pace. Cyber-criminals may use many attack vectors to acquire the victim's critical information to perform a fraudulent transactions, they may use False identities, stolen credentials, or social engineering to harm the users to their benefit. Consequently, one of the most important components of a corporate e-commerce platform is a secure payment processing system. Maintaining the trust of the customer depends on securing and protecting sensitive user and financial data, making this a vital component of business operations.

This study explores the technical and procedural components that lead to a robust and dependable system by examining best practices and techniques used in development of secure payment processing system and payment gateway integration of enterprise e-commerce platforms.

1.1.1 E-commerce Payment Processing

E-commerce (electronic commerce) is the activity of electronically buying or selling products on online services or over the Internet. Technologies like online transaction processing, supply chain management, mobile commerce, electronic money transfers, Internet marketing, electronic data interchange, inventory management systems, and automated data collection systems are all used in e-commerce. [2]

E-commerce payment processing refers to the system that is responsible for processing digital transactions i.e. accepting and processing digital payments for goods and services within an e-commerce platform. It includes the secure flow of transaction information between customers, merchants, and payment service providers. The processor is responsible for authorizing, capturing, and settling money for each online transaction.

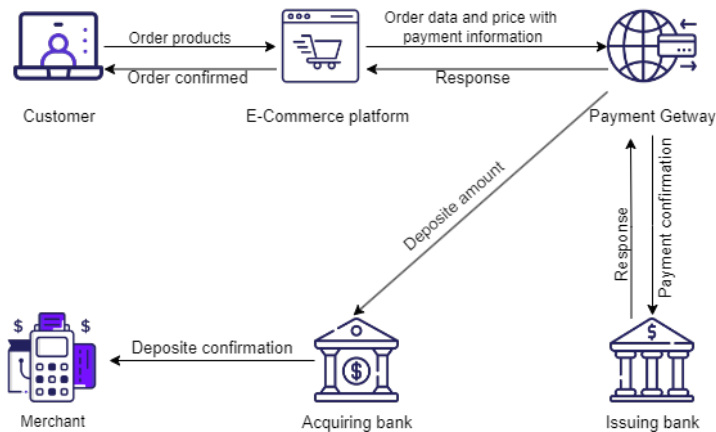


Figure 1.1: Example of an E-commerce Payment Process

1.1.2 Security Risks of E-commerce Payment Systems

The process of making a secure system starts by analysing the the security risks of the said system, consequently, developing mechanisms to counter the them. [3] This identification of the system's critical (e.g payment processing systems) components and the attacks that they are vulnerable to. Hence, of potential security risks of payment system of any e-commerce platform is very important. There can be many security risks involved in the said system. Some of the most common security risks (security attacks) are discussed below.

1.1.2.1 Malware

Malware refers to malicious programs such as viruses, trojan horses, ransomware and remote administration tools (RAT). Hacks may find backdoor in a payment systems to plant some malware to monitor, hijack, gain sensitive information, or perform unauthorized actions which is otherwise performed by administrator of the system. They can also plant malware onto an user's personal device through a compromised system, under control of a hacker, and use the device of the victims device to cause harm to the user and widen their network of compromised devices, increasing their size of **botnet** [4] as a result.

1.1.2.2 Man-in-the-middle attacks

A "man-in-the-middle" (MITM) attack is a type of cyber attack when an attacker silently transmits, eavesdrops, and potentially alters communications between two parties that assume they are speaking with each other directly as the attacker has already established itself between the two parties as a middleman and each communication passes through the middleman. Hence, it gives power to the attacker to listen, to sensitive communication retrieving critical information such as user-id, password, banking information and other, the attacker can also alter the communication between to parties.

For example, if a sender tries sends some money to a receiver; the attacker can intercept the traffic to alter the destination address to send the money to some other destination while sending two forged responses to both parties confirming successful transaction.

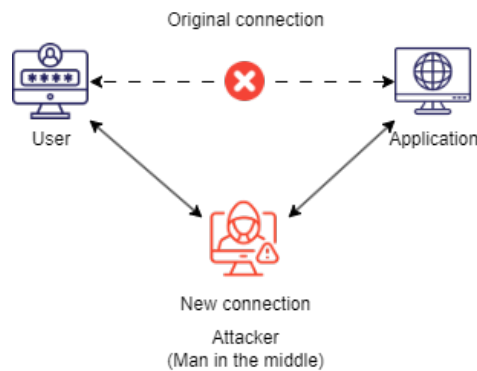


Figure 1.2: Man in the middle attack

1.1.2.3 Distributed Denial of Service (DDoS) Attacks

Distributed Denial of service attack refer to a type of cyber attack where the attacker or a group of attacker tries to overwhelm the capacity of a web server of processing incoming requests by flooding the server with superfluous requests from multiple devices (usually through a botnet of computers) which causes the server to slow down and disrupts the services of hosts connected by the network.

1.1.2.4 Remote Code Execution

Remote code execution (RCE) is a category of cyber attack where where the attacker can execute any commands into a target machine remotely. This is serious vulnerability and results of exploitation of this can be severe, for example the attacker can spawn a reverse-shell by using RCE of a vulnerable component in the application gaining low privileged access of the operating system controlling the application, the attacker can exploit other attack vectors after gaining the access of the system to gain highest privileges of the system id proper security measure are not in place at Operating system level e.g kernel exploits and other privilege escalation techniques.

1.1.2.5 Cross site Scripting (XSS) attacks

Cross-site scripting (XSS) is type of cyber attack where the attacker injects malicious scripts into the code of the trusted web application, as a results all users that user the application becomes affected by the malicious script; the may inject script that tell the user to go the an another malicious link which can be used to plan malware into the victims device or to perform CSRF (cross-site request forgery attacks) and even the it can be used to hijack the users session which can be used to perform malicious request on the users behalf (called session hijacking attack).

1.1.2.6 SQL Injection

SQL injection attack is the process of inserting, or "injecting," a SQL query into the application through client input data. When a SQL injection exploit is successful, it can read sensitive data from the database, alter database data (Insert/Update/Delete), perform database administration tasks (like stopping the DBMS), retrieve the contents of a specific file that is located on the DBMS file system, and occasionally even send commands to the operating system. One kind of injection attack is SQL injection, when preset SQL commands are executed differently by injecting SQL commands into data-plane input. [5]

1.2 Motivation

Numerous studies have been done on online payment processing methods, payment gateways, and cybersecurity. Nevertheless, the integration of various crucial systems in an application infrastructure poses multiple challenges and vulnerabilities in the platform. The principal reason behind this is the complexity, distinctiveness, and scale of a modern enterprise e-commerce platform.

Through years of research done in the field of the security of applications, the baseline of the most common security risks of an application has been established e.g **OWASP Top 10** [6], but the cause of these security risks tends to change from platform to platform. Hence, it is essential that security assessment is performed and best practices are followed during the development of an enterprise application as the vulnerabilities introduced in a system are dynamic and subject to change and the cause is often as small as a simple outdated software package or a mis-configured Access Control policy. These vulnerabilities introduce many attack vectors that can be creatively exploited by attackers to gain sensitive information about the system or to gain access of the said system. Naturally, the payment processing service of any platform is a prime target of an attacker, because it is one of the most critical parts of an e-commerce platform or a SaaS application. So, the security of the service becomes a necessity.

Therefore, along with automated methods, we need human intervention to perform security assessments throughout the software development life cycle of a platform for achieving the desired results.

1.3 Scope of Current Work

The major focus of this project are the development, gateway integration, security risk assessment, vulnerability mitigation of a business e-commerce platform's payment service. A live payment service is developed and integrated into a business e-commerce web application in order to accomplish the same goal. After that, Security assessment of the system is conducted via **grey-box testing** [7] using both automated and manual methods. Potential security risks are identified and demonstrated. Vulnerable components causing the security risk are identified, revised and re-developed in order to mitigate the attack vectors. And this process is repeated until all current vulnerabilities are mitigated.

1.4 Survey of Related Works

This section discusses different studies done the field of Digital Payment transaction methods, Payment gateway, survey of e-commerce security risks and cyber-security which are introduced by different researchers by their efforts. This survey study provides a brief summary of the study along with an analysis and evaluation of those approaches based on their performance criteria.

In his study [8], Lawrence et al. uses relevant annual reports disclosed by eBay (parent of PayPal), along with other eBay and PayPal documents, as a baseline and discuss about the potential risks factors associated with e-commerce, cyber, IT and online payment processing systems.

SH Chun, in his study, discusses about the the liabilities issues in electronic transaction systems when security breaches takes place. It reveals that firms can impose liability when the risk of breaches is high and the customer's loss is significant. However, liability depends on the customer's risk attitude, losses, and importantly security investment of the platform. [9]

H Nahari et al, in the book "Web Commerce security Design and Development" provides a comprehensive analysis of system security from holistic and atomic perspectives, defining risk driven security, protection mechanisms, and best deployment methods, focusing on e commerce and m-commerce, with a user-friendly implementation approach and also proposes techniques to implement an adaptive, risk-driven, and scalable security infrastructure and methods to identify weak security in a large-scale, transactional systems.[3] In this book they also discuss about most common vulnerabilities of an e-commerce transactional systems, and provides guidelines for implementation of security driven architecture to minimize security risks.

This research [10] study examines the development of e-payment systems, as well as the many traditional e-payment modalities. Additionally reveals a lack of solution techniques and security provisions. The primary objective of this survey study is to highlight the current state of digital e-payment systems and the prospects they bring for e-commerce systems of the future. This study also examined the fraudulent transaction rates that will for determining a benchmark for development of secure e-payment systems.

In this paper, Niranjnamurthy M. et al. included an overview of e-commerce security, describes the steps involved in placing a purchase online, explores different security issues in e-commerce, and proposes guidelines for a secure e-commerce process. [11]

The authors of [12] and [13] provided solutions for the user's anonymity issues

and highlighted security flaws. The rapid adoption of e-payments via cell phones and other mobile devices has made the user experience of digital transactions easier and more convenient. [10]

Francisco Javier Miranda et al. introduce a comprehensive Web Assessment index focusing on four categories: accessibility, speed, navigation-ability, and content. This methodology is equally valuable in analyzing various organizations for example commercial, educational NGOs, etc. To support the validity of the index, the proposed mechanism is performed on the websites of the top 200 Spanish firms. The evidence from the retrieved from the survey provided valuable insights in support of the proposed assessment mechanism. [14]

Siamak Solat et al, in this study, describes that as digital payment systems develop and incorporate strong levels of security strategies, in parallel the attack strategies of attackers become as advanced and creative as the security policies in place. The survey also aimed to identify and analyze both attack strategies and security solutions on the dominant electronic payment systems, focusing on the most successful attacks against them.[15]

S Nambiar et al. explores the security risks involved with digital payments in m-commerce. The study reviews the public key infrastructure as the baseline for security in mobile technology, explores mobile commerce and digital payments, and surveys the use of security technologies like WAP, SIM application toolkit, and J2M in the current m-commerce sector evaluating the performance of these security technologies for promoting a safe digital payment.[16]

1.5 Organization of the Report

In **Chapter 1**, the basic introduction of the project topic i.e e-commerce and associated security risks are discussed. The background related to the study is also discussed along with the current scope of the study. Literature Survey of related works in this topic is also discussed in this chapter.

In **Chapter 2**, basic theoretical concepts related to security analysis, security assessment tools, concepts related to web system exploitation, web application development, and some basic networking concepts are briefly discussed.

In **Chapter 3**, Approach to development of the system and process of integration of payment gateway is discussed.

In **Chapter 4**, The security Assessment process of the payment system is discussed. The security assessment results with risk implications to the system are analysed and presented.

In **Chapter 5**, Some recommended procedures and practices to mitigate the security risks are discussed and measures taken to mitigate the issues in the system are also discussed.

In **Chapter 6**, the overall work related to its advantage, shortcomings, future scopes, and conclusion are discussed.

CHAPTER 2

Practical Concepts and Approaches to Development

This chapter provides an overview of the basic practical concepts utilized in this study. The concepts of security analysis of web applications, security assessment tools, concepts related to web system exploitation, software and technology requirements for web application development are discussed.

2.1 Web Applications

Web applications are a type of applications which is served over world wide web (www), stored in a remote server, and accessed through web browsers. Web commerce relies on Web server's capacity to provide user-interactive apps. The usage of general purpose languages like PHP (Hypertext Preprocessor) and ASP.NET allows for easy development of dynamic Web pages, enabling increased interactivity.

2.1.1 Web Application Architectures

One of the most crucial part of web application design is to prioritize scalability for growing user base. Scalable architectures usually consist of multi-tiered modules that can be reused and for different use cases. There are many approaches to the development of a web app, but one of the most commonly used architecture for development of web applications is Model View Controller architecture (MVC).

2.1.1.1 Model View Controller Architecture (MVC)

In this architecture, there are three different components, Model, view and controller. This design pattern segregates the application into three interconnected logical components, i.e model, view and controller.

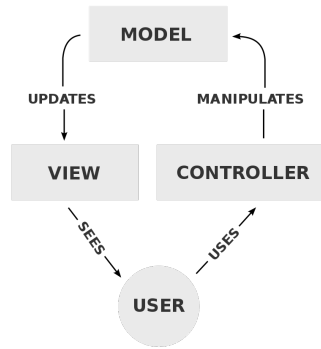


Figure 2.1: Model-View-Controller (MVC) architectural pattern [Image courtesy: Wikipedia, "Model-view-controller"]

- **Model:** This component is responsible for interacting with business-logic, and data. It manages the application's data, processes business rules and interacts with database through queries to perform business operations. Any model calls to the data store should be accomplished through secure means to prevent SQL injection attacks.
- **View:** Represents the User interface of the applications, it takes data from model component and generates user interfaces for the user based on user input.
- **Controller:** The Controller serves as a bridge between the Model and the View components. It accepts user input and modifies the Model as needed, as well as updating the View to reflect Model changes. It comprises application logic, including input validation and data manipulation.

The Model view controller architecture process is illustrated in Figure 2.2 as an e-commerce transaction.

2.1.2 Web Development Technologies

Developing a complicated web application requires technologies from both the server and client sides. Javascript is the primary client-side web language along with html and css, and it is used on the majority of online sites and web applications.

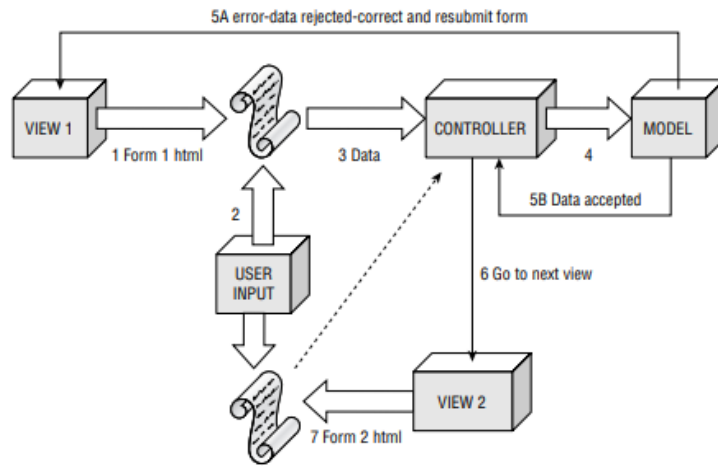


Figure 2.2: E-commerce transaction through MVC process [Image courtesy: H. Nahari and R. L. Krutz, Web commerce security: design and development.]

Hence most client-side frameworks and libraries are based on JavaScript.

For example some of the most commonly used modern client-side frameworks and libraries includes **React.js**, Angular.js, Vew.js, **Jquery**, Alpine with some exception begin blazor (based on **C#**), Web Assembly (based on rust) etc.

Although server-side technologies vary. Server side languages may be based on JavaScript as well e.g **Node.js**, Express.js, Nest.js among many others, but there are different technologies and frameworks that is not based on JS. The main reason is the single threaded architecture of JavaScript, it is generally slow for compute intensive task that needs more CPU and memory hence, **Java** and Java based frameworks are mainly used in enterprise web applications along with other languages like **Go**, **python**, **C#** etc.

Organizations need to choose a compatible database for web apps based on the different requirements throughout the web app development cycle, as well as the capacity to satisfy storage requirements and access speed. There are several types of databases available: Relational databases for example **MySQL**, Postgres, MSSql etc, Non-relational databases ex. **MongoDb**, In-memory databases etc ex. **Redis**.

2.2 Web-Application Security Testing

Security testing is a process of assessing the security of a network or system component by analyzing the effectiveness of its security points. This is done by analyzing the system's performance against the most common attacks that can be launched against similar systems. Security testing of web applications focuses on the analysis of different components of the application for any weakness, technical

flows, bugs that can be exploited, vulnerabilities, or a novel vulnerability. The security issues that are discovered are presented to the system administrator that contains the assessment of impact and potential mitigation techniques or a technical solution.

2.2.1 Types of Security Testing

There Are different types of security testing of web applications. The aim of all these testing is to ensure safety of the application form internal and external threats. However, any application is a subject to security threats at any point of time in-spite of proper security policies in place. Hence, several security testing methodologies provide a trustworthy way for enterprises to improve their cyber-security posture.

Vulnerability Scanning Test: This test examines a system components or network device, such as servers, routers, and endpoints, for potential security flaws in the network or compromised devices. This type of Scanning is automated and can also be scheduled weekly, monthly, quarterly depending on the organization. SISA is a Qualified Security Assessor (QSA) by PSI SSC that provides vulnerability assessment solutions from automation techniques. [17]

Security Scanning: Security scanning is type of security testing in which vulnerabilities in a network, web-server or a computer are examined through scanning the network, operating systems, applications, and web servers. Security scanning commonly involves searching for network and system weaknesses and identifying strategies to mitigate these risks. This is using both automatic and manual scanning techniques.

Penetration Testing: A penetration test, often known as a pen-test, is an authorized simulated cyberattack on a computer system performed to examine the system's security; this is not the same as a vulnerability assessment. The test is used to discover weaknesses (or vulnerabilities), such as the possibility of unauthorized parties gaining access to the system's features and data, as well as strengths, allowing for a comprehensive risk assessment to be conducted. [18] A common distinction between vulnerability assessment and penetration testing is that vulnerability scans are generally automated, whereas penetration testing is performed manually by a security specialist.

2.3 Security Testing and Assessment Tools

Security assessment and testing tools function to safeguard applications from internal or external security threats by identifying vulnerabilities and threats in an organization's network, cloud, applications, or database. Some functions of these tools include Vulnerability network security assessment, penetration testing, database security assessment among many others. Some of these tools are discussed below.

2.3.1 Burp Suite

Burp Suite is a software security application used for penetration testing of web applications. One of Burp Suite's key features is the ability to intercept HTTP requests. With this tool, HTTP requests can be routed directly from sender to Burp Suite, which intercepts the traffic and then to receiver acting as a proxy, or "man in the middle," between sender and the web service and receiver, allowing more precise control over the traffic transmitted between tow parties. With addition to proxy, it features a vulnerability scanner, an intrusion tool (for crafting different attacks) and a repeater tool for repeated attacks.

2.3.2 Nmap

Nmap is a surveillance tool for network discovery and security audits. It includes ping scanning (determining which hosts are up), a variety of port scanning techniques, version detection (determining service protocols and application versions listening behind ports), and TCP/IP fingerprinting (identifying remote host OS or device). Nmap provides customizable target and port specifications, decoy/stealth scanning, sunRPC scanning, and more.

2.3.3 Wireshark

Wireshark is an open-source network packet analysis tool. It is primarily used for network troubleshooting, network traffic and packet analysis, software and communication protocol development. It provides real time view of network traffic allowing detection of anomalous traffic, real-time insights, and protocol based filtering and other analysis methods.

2.3.4 Nessus

Nessus is a remote security scanning tool that examines a computer and alerts you if it finds any vulnerabilities that malicious attackers could exploit to obtain access to any computer that is connected to a compromised network. It accomplishes this by doing over 1200 checks on a particular machine, and determining whether any of those attacks could be used to break into the computer or otherwise harm it. [19]

2.3.5 Metasploit

The Metasploit Framework is an open source framework for vulnerability discovery, exploit development, and building custom security tools. It can be used for development and execution of exploit code onto a remote target machine. Metasploit is most typically used by information security professionals for penetration testing to identify and remediate existing vulnerabilities across an organization's network.

2.4 Role of Cryptography in Web Security

Web commerce transactions include sensitive information, which if compromised can lead to identity theft, financial losses, reputational harm, unauthorized access, compliance concerns, and mitigating expenses. Cryptography is a tool that can be used to protect a person's private information. [3]

Cryptography aims to prevent unauthorized access to transmitted or stored information. Ideally, Unauthorized individuals are unable to decrypt an encrypted transmission. However, in theory, an encrypted transmission can be decrypted through un-indented or brute-force methods but the time taken to accomplish such task is so substantially large that it becomes impractical to attempt the same. Hence, It is the state of the art to encode sensitive messages through cryptographic algorithms before the transmission of the message.

There are two principle forms of cryptography: **symmetric key** (secret or private key) and **asymmetric key** (public key). In symmetric key cryptography, the receiver and sender share the same secret key. In asymmetric key cryptography, both sender and receiver have private keys that are kept secret and public keys that are open for communication. [20] [21] The public and private keys are mathematically related. In an ideal scenario, a person with the public key cannot derive the private key. Symmetric key cryptography is typically faster than asymmetric key

cryptography due to less processing required. [3]

2.4.1 Symmetric Key Cryptography System

In this type of cryptography, both the sender and receiver has access to a secret key. Using this secret key the message is encoded before transmission and decoded after receiving. The overall challenge is to make sure both party has secure access to the secret key. Secret key systems with significant key sizes (> 128 bits) are highly secure. These methods are fast and efficient for encrypting massive amounts of data.

A symmetric key system needs the sender and receiver to share the same secret key, resulting in the need for a unique key for each recipient. To communicate in secret with other participants, each individual need $\frac{n(n-1)}{2}$ keys. For increased security, the secret key should be changed at frequent intervals. Ideally, a particular secret key should only be used once. [3] Examples of symmetric key cryptography include AES, DES, and 3DES.

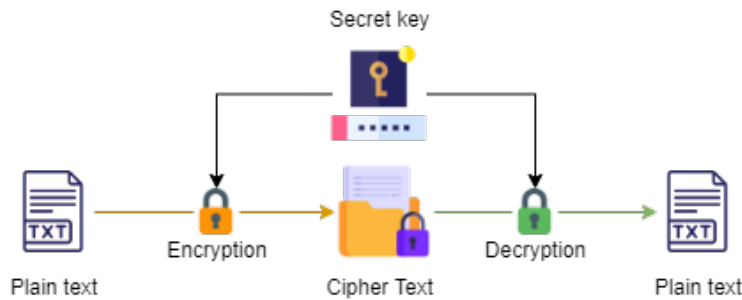


Figure 2.3: Symmetric key cryptographic system

2.4.2 Asymmetric Key Cryptography System

An asymmetric encryption technique utilizes two distinct keys, one private and one public. The public key is used for encryption and can be known by others, while the private key is used for decoding and must be kept secret. Like most public cryptosystems, asymmetric encryption is based on one-way mathematical functions.

This indicates that while it is simple to compute the result from given input data, it is difficult to recover the input data from the output. Furthermore, the underlying mathematical difficulties are thought to be difficult, making it computationally impractical to decrypt a communication without knowing the private key. [22] The following are the important points to note regarding public and private keys:

1. The public key cannot decipher the message it encrypts.
2. The private key is kept private.
3. Ideally, the private key cannot be inferred from the public key.
4. A message that is encrypted by one of the keys can be decrypted with the other key.

When Kp is the public key and Ks is the private key, the process is as follows [3]:

$$C = Kp(P) \text{ and } P = Ks(C)$$

Where C is the ciphertext and P is the plaintext. similarly the reverse is also valid.

$$C = Ks(P) \text{ and } P = Kp(C)$$

Examples of asymmetric key cryptography algorithms include RSA, Diffie-Hellman, and Elliptic Curve Cryptography (ECC) etc.

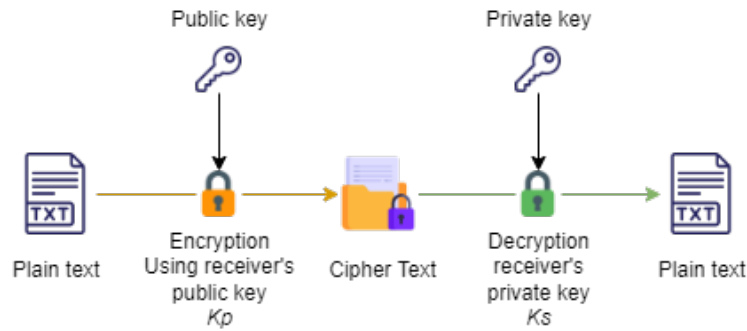


Figure 2.4: Asymmetric key cryptographic system

2.4.2.1 One Way Functions and Trapdoors

A one-way function is a function that is easy to compute in one direction, yet is difficult to compute in the reverse direction. For example, given $y = f(x)$ where y is the derived value from given input x . It is very easy to calculate y , given x but very difficult to derive the value of x from the output y . Public key encryption is achieved through the use of one way function.

By applying the concept of one way function, public key is derived from a private key, by applying a mathematical formula. But deriving the private key from a given public key is mathematically exhausting, hence very difficult to compute. i.e $pk = F(sk)$ where, pk is the public key generated from private key sk using mathematical function F .

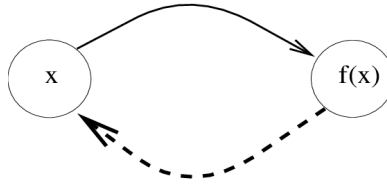


Figure 2.5: One way function

To be effective in public key cryptography, a one-way function must include a trap door. A trap door is a hidden mechanism that allows for one-way reversal capability. Knowing the trap door allows for easy derivation of x from given y in the previous example.

A **trapdoor** function is a special case of one-way function $F : X \rightarrow Y$ that contains a "trapdoor." The term "trapdoor" refers to specific information usually obtained during the process of creating a TOWF that can provide an easily computable inverse of a said function. In other words, whereas $f^{-1}(x)$ is considered to be unsolvable by modern computers, equation: $y = f^{-1}(x, t)$ (with t being additional knowledge) is easily calculated. [23]

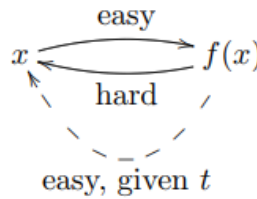


Figure 2.6: Trapdoor One way function

2.4.3 Digital Signature and Hashing

A digital signature, in cryptography is a mathematical system that verifies the authenticity of digital messages or documents. A genuine digital signature on a communication ensures that the message was sent by known and valid sender. Digital signatures in e-commerce can be used to prevent unwanted data modifications, authenticate signatories, and ensure non-repudiation.

The Digital signature of a message is calculated by passing the message through a **one-way hashing function** H which through cryptographic transformation produces a fixed length output, called the message digest.

After the message digest is calculated, this is encrypted with the sender's private key, this value is the signature of the message to be sent. During the transmission, this signature is sent with the message.

At the receiving end, the the message digest of the received message is calculated using the same hash function and the signature that is sent is decrypted using the sender's public key and both values are compared, if both are same, the the message is valid and it is not modified in transmit.

Hash digest of a message alone can be used to check the integrity of the message. For example, ideally for each message, the hash digest is unique and for each message that is same the hash digest is also same. Hence, the sender of the message can send the hash digest of the message which the receiver can check for integrity by comparing the sent hash and calculating the hash of the message. But in this case the hash of the message, can be altered in transmit along with the message, hence it goes back to the same issue of validity. Hence, the encryption of the hash is necessary to insure the safety of the message to be sent.

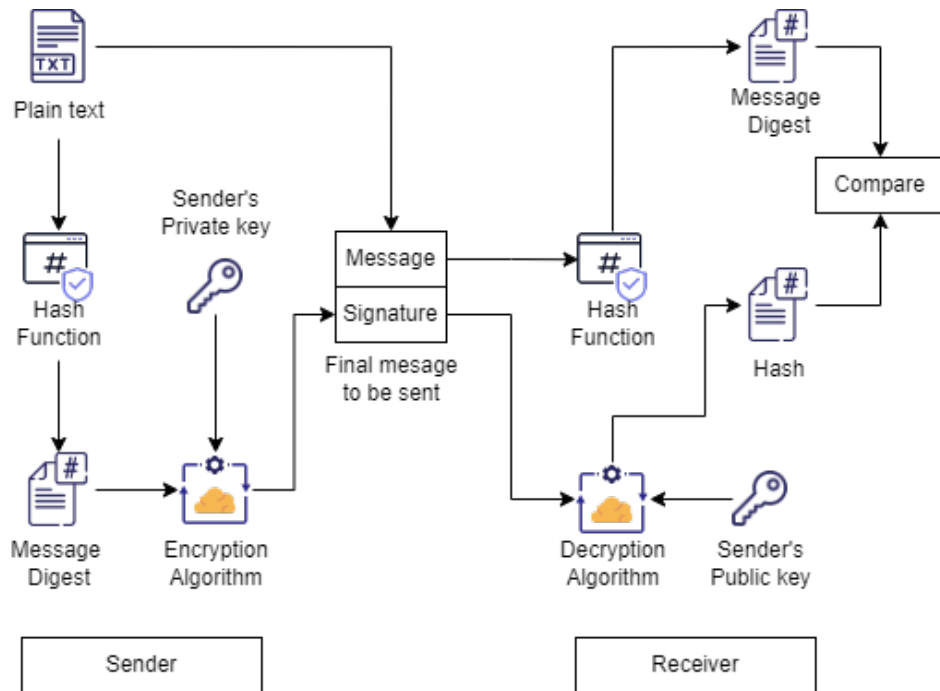


Figure 2.7: Process of digitally signing a message

2.4.3.1 Modern Hash Function Standards

This Standard specifies secure hash algorithms - SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 and SHA-512/256 - for computing a condensed representation of electronic data (message). When a message of any length less than 2^{64} bits (for SHA-1, SHA-224 and SHA-256) or less than 2^{128} bits (for SHA-384, SHA-512, SHA-512/224 and SHA-512/256) is input to a hash algorithm, the result is an output called a message digest. [24]

CHAPTER 3

System Development and Payment Gateway Integration

In this chapter, The process of development of the system, functional requirements along with the process of selecting and integrating the payment gateway into the payment system is discussed.

3.1 Development Process Overview

For this study, a live payment service for an enterprise e-commerce platform is developed, appropriate payment gateway service is identified and integrated into the payment processing service. Feature requirements are gathered and based of the requirements, the system is developed and integrated as part of the whole e-commerce platform.

3.1.1 Technologies and Tools Used in Development

In development of a web application service, a stack of various technologies and frameworks are used for building and integration and deployment of the final system, below is a list of technologies used in development of the payment service.

- Client-side i.e User Interfaces: HTML, CSS, JavaScript, JQuery, Bootstrap.
- Server-side i.e Backend: PHP
- Database: MySQL
- Server: Apache and AWS

The technologies, libraries and packages specified above are the requirements needed in development of the application. Below is a short description of the technologies used in development of the application.

HTML, or Hypertext Markup Language, is the industry standard markup language for documents displayed in a web browser. It describes the content and structure of web pages. It is frequently assisted by technologies like Cascading Style Sheets (CSS) a style sheet language used for styling of the web pages and scripting languages like JavaScript which provides user interactiveness and reactiveness of the web-page. JQuery is a JavaScript library designed to simplify HTML DOM tree traversal and manipulation, as well as event handling, CSS animations, and Ajax. [25] PHP is a general-purpose server-side scripting language that is widely used in web development.

PHP code is typically executed on a web server by a PHP interpreter that is implemented as a module, daemon, or CGI executable. On a web server, the result of the interpreted and executed, which can be any type of data, such as produced HTML or binary image data as a part of an HTTP response. [26] MySQL is an open-source relational database management system. It is one of the most commonly used relational databases management systems in the current times along with other RDBMS technologies such as PostgreSQL.

3.2 Functional Requirements

In software engineering, functional requirements describe the expected operations of a system and its components based on inputs. To define the functions of a system, the basic requirements must be defined. The environment in which the system performs its functionalities is such that:

1. As input, it takes a payment link generated from other services e.g Cart service or Order processing service of the commerce platform.
2. Based on the validity of the generated payment link, it performs the required payment processes.
3. On a successful or failure of payment it provides outputs required data for further processing of the specific order for which the payment link was generated.

Detailed functional requirements i.e description of the operations performed by the service is discussed below.

- The service should have a good user interface.

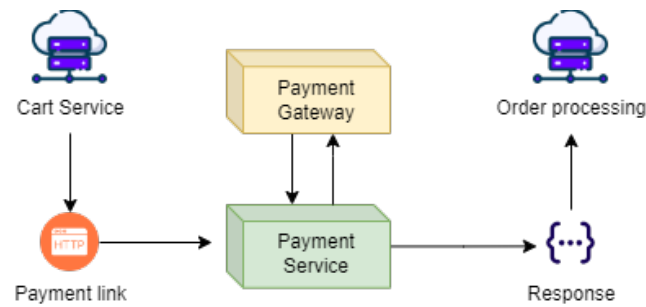


Figure 3.1: Diagram of payment service

- It should take payments links generated by other services as input.
- Using the information from payment link it should display order information and total price of the order.
- The Service should integrate with a payment gateway to process the payment for the order amount.
- It should be able to process the output generated from payment gateway to further generate structured response as final output which is to be used for further processing of the order (i.e dispatch of the order).
- The service should process data securely so that fraudulent transactions is minimized.
- It should process data and produce output(s) accurately.
- Most importantly, the service should be highly available and with minimal errors as the service will be frequently used by large number of users.

3.3 Payment Gateway Integration

A **payment gateway** is a software solution that allows online businesses to manage and authorize digital payments from customers such as debit or credit cards, digital wallets, and electronic bank transfers. Payment gateways are intermediate services that verify, accept, or reject digital transactions on behalf of online businesses. Some of most popular payment gateway solutions include: Razorpay, Paytm, PayPal, PayU, Stripe, OPEN, Bill desk, CCAvenue etc.

The integration of payment gateway requires various steps: Setup of merchant accounts for preferred service, Obtaining API keys, and other credentials to be used when integration with the platform, Integrate Payment gateway into the web-application or e-commerce application, Testing, and finally deployment of the whole system. Below are the brief description of each process:

- **Setup merchant account:** The first step of the whole process is to set up merchant account to receive payments from customers and register with the service to be used.
- **Get API keys:** In order to use the services offered by the payment gateway, it is essential to get API keys so that critical functions offered by the service can be used through the protected API endpoints. Along with API keys, other credentials may also need to be obtained which may vary.
- **Integration:** This step can differ based on the payment gateway and website platform. Some e-commerce platforms, such as Shopify and WooCommerce, provide plugins and extensions to make integration easier. Otherwise, it is needed to manually integrate the gateway service into the platform using the provided applications programming interfaces, Rest-API endpoints or software development kits.
- **Testing:** Most gateways include a sandbox or testing environment in which you test purchases can be made to verify that the system is working as intended and everything functions correctly.
- **Deployment:** Once testing is completed and everything works as intended, the system can be go live. Customers should be able to purchase and make payments for orders at that point.

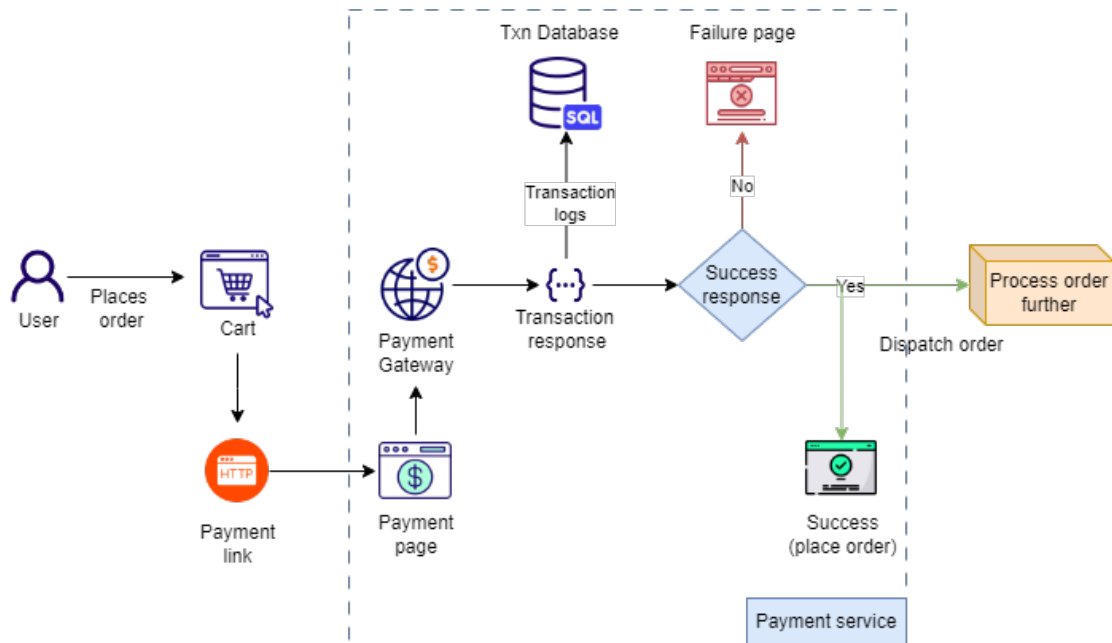


Figure 3.2: Process of online payment through payment service

Based on the given requirements and discussions with stakeholders, **PayU** [27] was selected as the payment gateway to be integrated with the payment service. All the

necessary steps were followed and required credentials were obtained to integrate the gateway into the platform. A detailed payment flow after payment gateway integration into the payment processing system is illustrated at figure 3.2.

Using the given requirements and the payment service architecture, the payment screen was developed. The main payment page is responsible for fetching order information from the payment link generated by cart service, displaying order information, i.e ordered product, detailed product description, user and dealer information, total payment to be paid, and various other options regarding ordering products for example tentative delivery date changing and others. Figure 3.3 displays the resulting payment screen that was developed.

The screenshot shows a web browser window with the URL `dev.payment-serv.com/payment/?f2+wQeExAzwyMi+31IWqsRqOoYnCwc8Yb7HgQRsfWpA=`. The page is titled "ORDER DETAILS" and contains the following sections:

- Booking Information:**
 - BOOKING ID: 00000056
 - DATE OF BOOKING: 2024-04-15
 - DATE OF PO: NA
- Customer Info:**
 - Customer Name: Adi Abdal
 - Email Address: tst.svixc10@gmail.com
 - Contact Number: 7917781579
 - Customer Address: Sarberia relaince petrol pump, South_24_Parganas, WEST BENGAL, India, 743329.
- Dealer Info:** (Empty)
- AMC Info:** (Empty)
- Price Details:**
 - Total Amount: Rs.171115 (Price includes 18% GST)
 - Product Amount: Rs.147125.00
 - AMC Charges (ExtendedRs.23990 Warranty)
 - Amount to be paid: Rs.97552.50 (50.00% of Product Amount + 100% of AMC Amount)
 - PAY NOW** button
 - *Inclusive of installation charges of INR 2301(Inclusive of GST) for Doors & INR 1770(Inclusive of GST) for Windows & ventilator per unit
 - *Exclusive of payment gateway charges
- Product Information:**
 - Model Name: Classic
 - View Full Specifications -
 - Width x Height (in mm): 900X2050
 - Design Code: E01
 - Color Code and Name: B-02-01 - Lava Teak
 - Description: External Single Solid with 100 MM Frame
 - Architrave: NSF
 - Any Combination: With - NA

Figure 3.3: Main payment screen

At this page, total payable amount is displayed using the order info, this information is fetched using orderID obtained from the request params. Clicking pay now redirects the user to a screen where the user has the option to pay online (through payment gateway) or offline (i.e Pay on Delivery). Online payment redirects the user to the payment gateway where the user can pay the total amount using methods supported by the payment gateway and after successful or unsuccessful payment, based on the response, success or failure screen is displayed and further processing of the order is performed i.e order dispatch, email confirmation etc.

CHAPTER 4

Security Assessment and Results

In this chapter, The process of security analysis, security assessment frameworks and findings from the experiments are discussed. Potential security vulnerability exploitation processes and risk implications to the system are also discussed.

4.1 Security Assessment of Payment Service

After the development of the payment service was completed, to measure the security risk of the service security assessment of the service is performed as this service is to be used frequently and large transactions are to be performed, it is necessary that the security risks are minimal.

4.1.1 Security Assessment Approach

The approach to the assessment of the web application is divided into multiple steps, where at each steps various intimations about the system is discovered, vulnerable components are identified, level of threat to the whole system is assessed and finally possible mitigation processes are discussed. The steps followed in the assessment is discussed below:

- At first step, system to be assessed i.e the target is identified, for this use case, the target in which the testing is performed is a valid payment-link which is used for the testing of online payment through the system.
- The next step is to identity the architecture of the system i.e this the server in which it is running, the operating system, technologies used etc. From

this information exploitation planning can formed as different systems are vulnerable to different kinds of attacks, especially out dated systems.

- After that, The whole system is scanned for discovering of assets i.e all directories of the system or maybe hidden directories which is not meant to used by end-users or directories that used by admins using tools like nmap, dir-buster etc.
- The next steps to perform automatic vulnerability scans for OWASP Top 10, WASC vulnerability etc, using various security assessment software e.g burp suite, ZAP etc.
- The next step is analyse the automated scans and check for severity the found threats, performing manual operations to simulate common attack, and observe the affects.
- The final step is to assess and document the the level of threat for each vulnerabilities found and possibly discuss mitigation techniques suppressing the same.

4.2 Security Assessment Results

After Manual and automated testing using security assessment and pen testing tools, some vulnerabilities in the payment system were discovered. The results, along with vulnerability exploitation methods, possible risk implications to the systems are discussed in the following sub sections.

4.2.1 Manipulation of Payment Using Parameter Tampering

During manual pen testing, it was noticed that an attacker can intercept, the request sent to, or the response send by, the payment gateway. In this case, the attacker can be a middle man between request or response and the destination. Therefore, can tamper with the request and response. This type of attack is known as **Man in the middle attack** (MITM) refer to 1.1.2.2.

4.2.1.1 Risk Description

Using this type of attack, **an attacker can get an order to dispatch without paying** by following the process described below:

1. The attacker adds desired products into cart. The cart service generate payment like.

2. The attacker then selects online payment, and is redirected into payment gateway page, hence a transaction is started by the payment gateway.
3. After that, the attacker cancels the transaction from the gateway page at the same time intercepts the response sent by gateway.
4. The gateway sends transaction failure response due to canceled transaction, but the attacker tampers with the JSON response sent by the gateway, and changes the JSON payload identical to a success response sent by gateway (e.g successful transaction), while keeping other order information same for example, order-id, product-information, etc.
5. After the changes is made to the response, the attacker forwards the tampered response onto the intended destination (i.e a success response). which causes Success page to be rendered on to the user/attacker screen and trigger the further processing of the order e.g dispatch of order and email confirmation etc.

In summary, an attacker can exploit this vulnerability to order products without payment by intercepting and tampering the request and response generated by the payment system. Using the same the attacker can place multiple orders and there for, it can greatly impact the platform financially. Figure 4.4 describes this attack process for this use case.

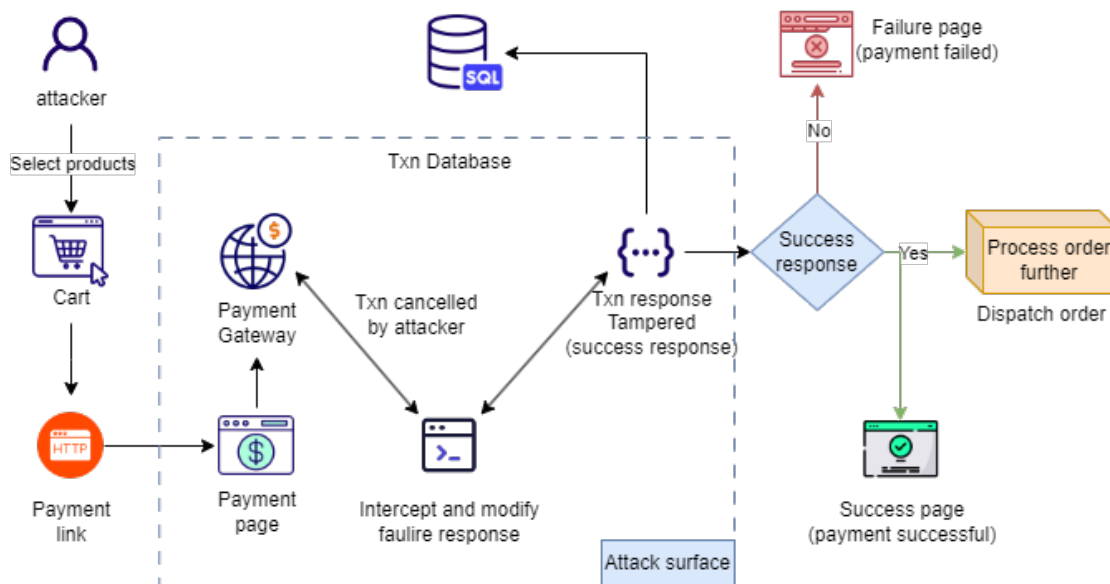


Figure 4.1: Parameter tampering attack

4.2.2 Exposure of critical Information Using Insecure Direct Object Reference (IDOR)

Insecure Direct Object Reference (IDOR) is a vulnerability in which attackers can access or modify objects by manipulating identifiers in a web application's URLs or parameters. It arises as a result of missing access control checks, which fail to verify whether a user is authorized to view certain data. When effectively utilised, using this vulnerability, the attacker can get access to critical information by manipulating the object. [28] Example process of IDOR is described in figure 4.7.

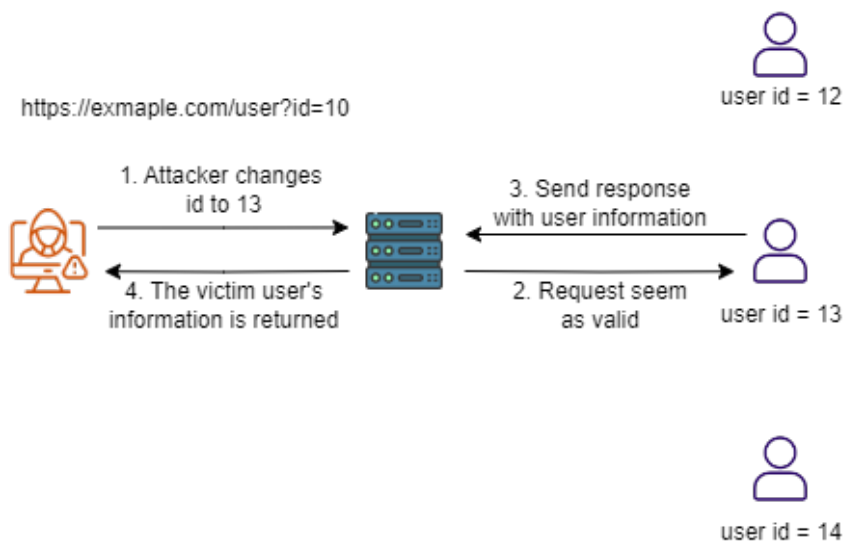


Figure 4.2: Insecure direct object reference attack example

In this case, it was observed that an attacker can use IDOR vulnerability on render time of the main payment screen and get critical information using the order id from the request parameter. The attacker can inject arbitrary order id to get personal information of customers, dealer information i.e email addresses, phone numbers among others, causing potential leakage of critical user information which may be used by other attacker for constructing different attacks.

4.2.2.1 Risk Description

Using this vulnerability, the attacker access critical information of different order bypassing access control checks put in place e.g access user information or order information etc, by injecting arbitrary order id in the request parameter. An example of this attack is discussed below:

1. The attacker gets access to a payment link.

2. The attacker intercepts the request before it hits the server, the page is rendered.
3. The attacker manipulates the parameters of the request and injects an arbitrary orderId.
4. The page is rendered against the order id sent with the request and the attacker gets access to confidential information.

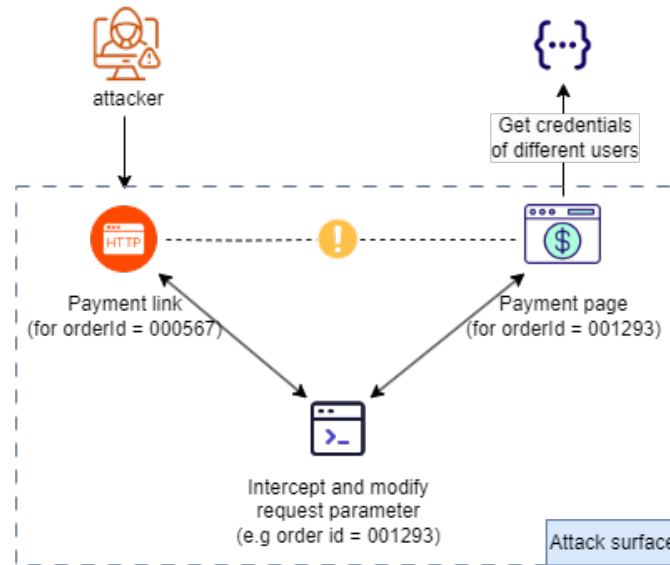


Figure 4.3: IDOR attack process of this case

4.2.3 HTML Injection Attack

HTML injection is a type of code injection vulnerability that occurs when a user is able to control an input point and is able to inject arbitrary HTML code into a vulnerable web page. [29] This vulnerability appears when user input is not sanitized and the output is not encoded (e.g HTML tags are not encoded). HTML injection enables the attacker to embed malicious HTML code snippet into the vulnerable page. The targeted browser will be unable to distinguish (trust) genuine from malicious elements of the page, and as a result, it will parse and execute the entire page including the malicious HTML code within the victim's context.

4.2.3.1 Risk Description

For this case, the vulnerable component is the success or failure screen of the payment system. The attacker can inject malicious code snippet on top of the code of the page at render time of the page. Using this attack the attacker can execute

code refer to XSS 1.1.2.5 that can steal the victim user's credential for example session tokens, Json web token and among many other information if not properly secure. The process of the exploit for this use case is discussed below:

1. The attacker can modify the request URL parameter of the success or failure page.
2. Take a url `http://exmaple.com/success/orderId=000112`; here the attacker can insert html code on to the request url that is for exmaple:
`http://exmaple.com/success/orderId=<h1>000112</h1>`
3. Although this html injection can be seem harmless but, code can be injected here such that it executes some JS code that can steal the session token, or access token of who ever opens the page.

4.2.4 Insecure Cookie Attribute Configuration

It is a recommended practice that secure and HTTP only attribute is set by the application server when sending a new cookie to the user within an HTTP Response. HTTP-Only attribute ensures that client-side JavaScript cannot access the cookie value from browser. The secure attribute's goal is to prevent cookies from being seen by unauthorized parties because they are transmitted in clear text and Secure attribute ensures that the cookie is sent through HTTPS resulting the transmission is encrypted hence is cannot be seen by unauthorized parties. It is the recommended practice when development of web application that relies on cookies for different functions, but it often overlooked, which is observed in this case as well.

4.2.4.1 Risk Description

It was observed that in the application, the session cookie and access-token-cookie generated by the application, has the HTTP-only and secure as false, this causes the cookies to be exposed by client-side JavaScript and the cookies are transmitted as plain-text as secure attribute is not set. This escalates the risk of session-hijacking attack and potential exposure of critical credential e.g access-token used by the application that is needed for calling other services such as order processing service.

An attacker can exploit some attacks such as HTML injection (which this applications is vulnerable to as well) or XSS attack and plant some malicious code that can fetch cookies set at victim's browser with no HTTP-only and Secure flags.

This code is executed when the user opens the infected page and cookies with

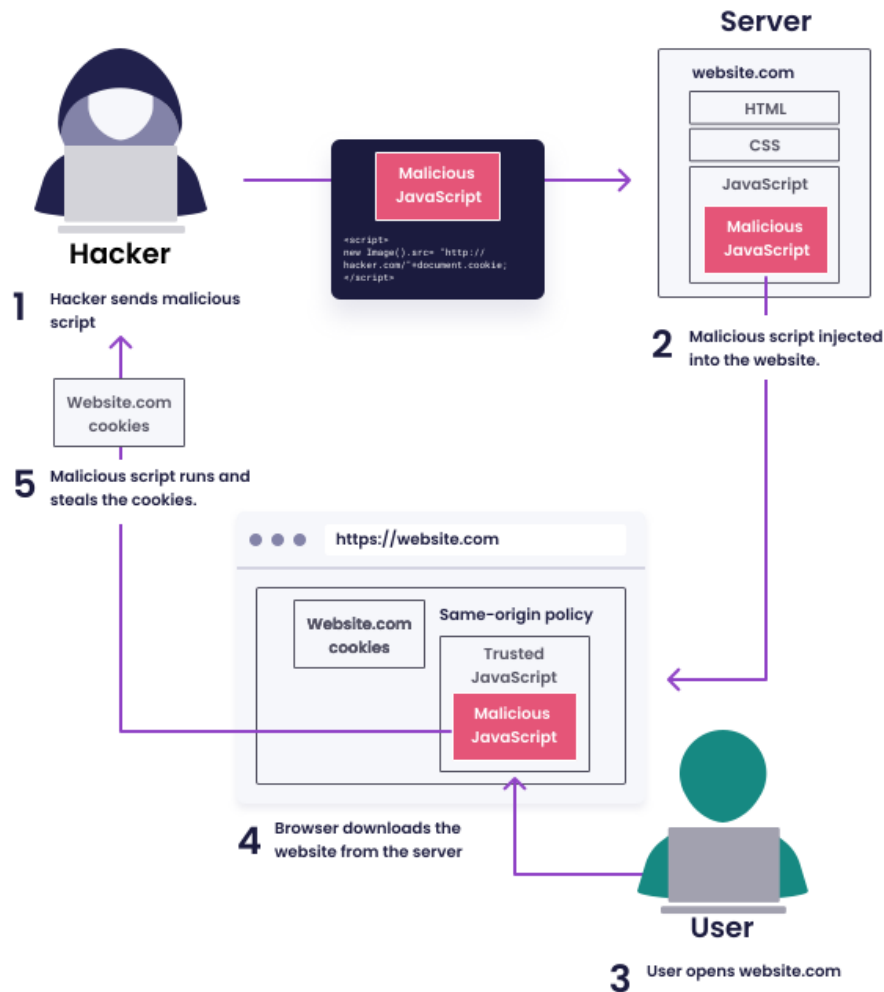


Figure 4.4: A Code injection attack escalating to an XSS attack [Image courtesy: Cross-site scripting (XSS) by Snyk Learn, url: <https://learn.snyk.io/>]

mis-configured flags and the attacker gets access to session cookie. Now the attacker can use the session cookie and use the cookie to hijack the the victim's session and perform unauthorized transactions.

4.2.5 CORS Misconfiguration

Cross-origin resource sharing (CORS) is a mechanism that allows a web page to access restricted resources from a server on a domain different than the domain that served the web page. [30]. It allows a server to specify any other origins (domain, scheme, or port) from which an application may send a request and the server may respond with data. CORS also relies on a technique that allows browsers to send a "preflight" request to the server hosting the cross-origin resource to ensure

that the server will allow the subsequent request. During the preflight, the browser transmits headers indicating the HTTP method and headers that will be used in the real request. At the server configuration, it is to be specified that from which domains the request for resources can be made by using **Access-Control-Allow-Origin** flag. Access-Control-Allow-Origin is a response header used by a server to indicate which domains are allowed to read the response. Based on the CORS W3 Specification it is up to the client to determine and enforce the restriction of whether the client has access to the response data based on this header. [31]

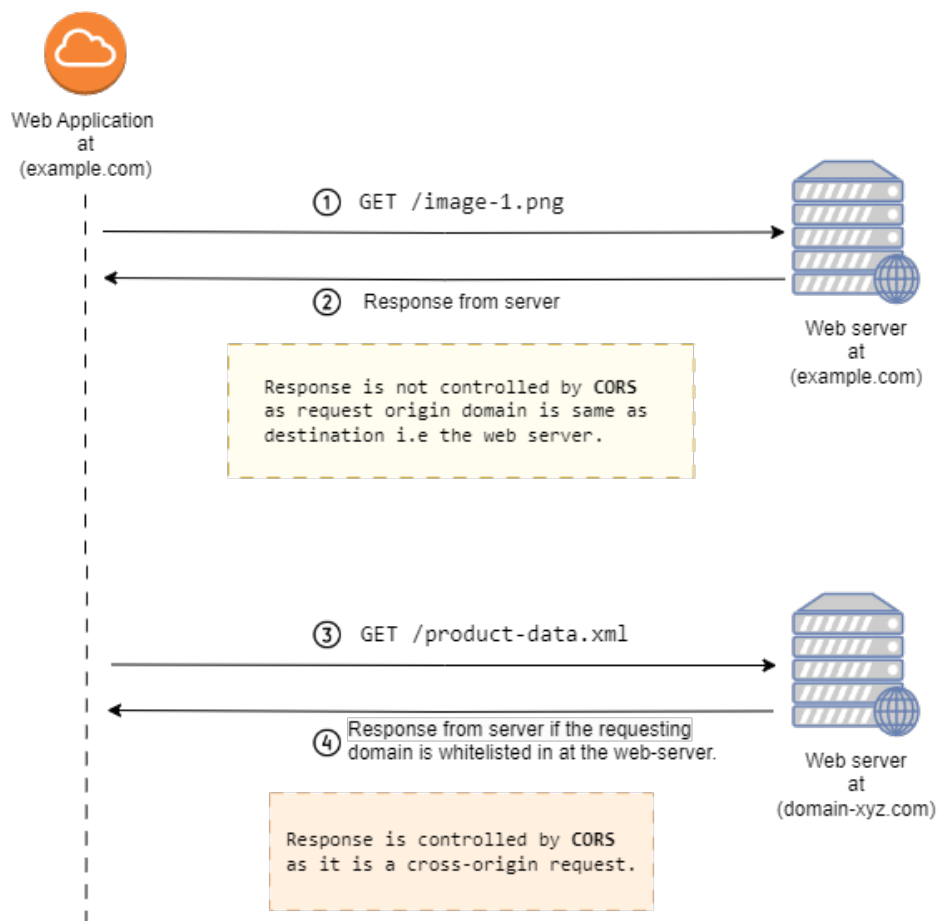


Figure 4.5: Cross-origin requests controlled by CORS configuration

From a security testing standpoint it is important to search for unsafe setups, for example applying a * wildcard as value of the **Access-Control-Allow-Origin** header that indicates all domains are permitted. Another insecure example is when the server returns the origin header without performing any additional checks, allowing access to sensitive data. It should be noted that enabling cross-origin queries is extremely unsafe and is not acceptable in general, with the exception of a public API that is designed to be available to anyone.

4.2.5.1 Risk Description

Misconfiguration of CORS settings at a web-server, can lead to some serious issues. Attackers can exploit CORS misconfiguration vulnerability to perform various attacks into the web-server. For example An attacker can send requests from any random origin to the server.

In this case an attacker can send a malicious request (i.e get user-data) via a web URL in control of the attacker and capture the response sent by the destination web-server. This way the attacker can access the responses sent by the compromised web-app through a 3rd party malicious Web-application.

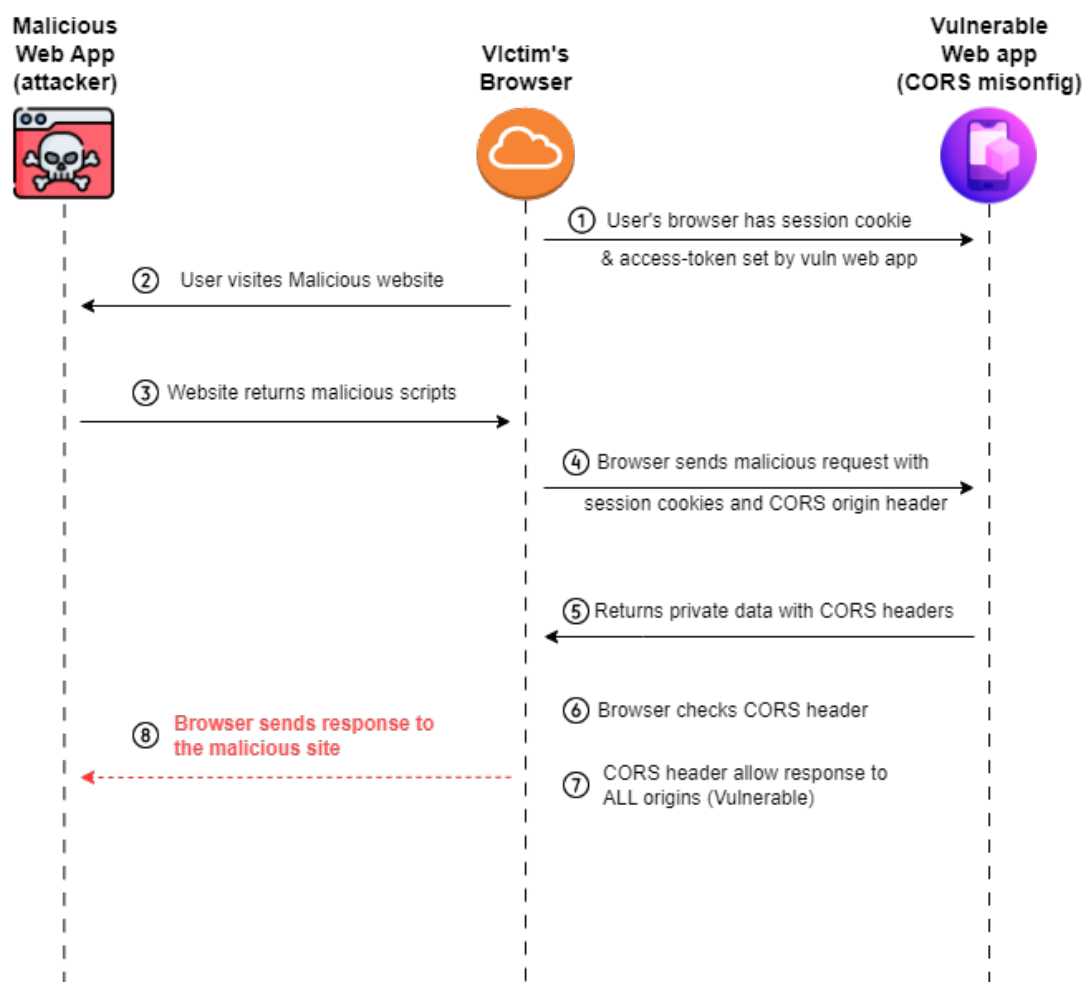


Figure 4.6: Exmample of a CORS misconfiguration exploitation [Image courtesy: D. Danelon, The complete guide to cors (in)security]

Using the information, CORS attack, the attacker can further escalate attack vector by using captured session tokens, access-tokens and perform CSRF [32] attacks on to some critical end-points for-example calling order-processing endpoint or calling payment end-point for an arbitrary order-id generated by attacker and many others.

CHAPTER 5

Vulnerability Mitigation Procedures

In this chapter, The recommended mitigation measures, practices for the discovered vulnerabilities in the system are discussed. The measures taken against the security issues are also presented in this chapter.

5.1 Vulnerability Mitigation Process Overview

Once a vulnerability has been identified , the following step is to prioritize how to manage that vulnerability with the original stakeholders in the organization or network. There are several approaches to treating vulnerabilities, such as:

- **Remediation:** Completely correcting or patching a vulnerability so that it cannot be exploited. This is the optimal treatment choice.
- **Mitigation:** Reducing the chance and/or severity of a vulnerability being exploited by applying proper measures and best practices.
- **Acceptance:** Taking no action to fix or mitigate the likelihood/impact of a vulnerability being exploited. This is often justified when a vulnerability is considered a low risk, and the cost of addressing the vulnerability is much larger than the cost sustained by an organization if the vulnerability was exploited.

For this case, The whole payment system of the e-commerce applications is assessed for security vulnerabilities, some vulnerabilities are discovered, following the recommended security assessment processes and manuals. These are discussed in the previous sections e.g 4.2.

5.2 Mitigation of Discovered Vulnerabilities

After Manual and automated testing, using the recommended procedures the discovered vulnerabilities which are presented in the previous sections. In this section are recommenced procedures to mitigated the found vulnerabilities are discussed along with the measures taken in this case to mitigate the security issues. The recommended mitigation practices and mitigation measures taken against the security issues are discussed in the following.

5.2.1 Manipulation of Payment Using Parameter Tampering

For this case, it was noticed after initial assessment that the system is vulnerable to Manipulation of Payment Using Parameter Tampering which is kind of MITM attack in which an attacker can intercept, the request sent to, or the response send by, the payment gateway. The detailed process of exploiting this vulnerability is discussed in the previous sections refer to 4.2.1.1.

5.2.1.1 Possible Mitigation Measures

One of fist measure that could be taken in order to mitigate this type of attack, ensuring that the information exchange is happening in a secure environment i.e a network that is not compromised. Also making sure that the request and response are begin taken place in a **HTTPS** connection Because HTTP connection is highly non-secure as communication is taken place in plain text where as in HTTPS communication is encrypted using TLS/SSL certificate, hence any one can see the requests and responses, hence critical information such as username, password, banking information etc can be exposed. Hence making sure that communication in secure network is one most crucial step to mitigate this type of attack.

Another Approach that can be taken is to mitigate this attack is that the payment gate-way response should be encrypted and after it reaches the destination, it is decrypted by the server and then based on the decrypted information, further steps taken i.e dispatch of order and proper response page rendering etc. In this way the middle man will be unable to tamper with the response.

To mitigate this type of attack, another measure that could be taken is that, for each request or response, the hash of the payload of the request or response should be sent. Each request sent by any client should be check for integrity by the server using the hash sent with request and some other secret key shared by client and server. And by the same way, each response sent by the server should be check for

integrity by the client using the hash sent with the request.

5.2.1.2 Mitigation Measures Taken

In order to mitigate this issue in the system, measure taken was verifying the request and response integrity between the client and server communication, in this case between the back-end of the system and the payment gateway service. The detailed process followed in collaboration with the gateway service provider (**PayU**) is discussed below:

In this use case both the back-end and the gateway provider shares a **Key** which is used to uniquely identify each merchant and **Salt** which is used as a secret for calculating hash of payload to be sent or received.

For each request sent to the payment gateway the process is discussed below:

1. With the payload to be sent to the gateway as request, the hash calculated from the payload is sent with the request as well.
2. The formula for calculating the hash is done using SHA512 algorithm using a formula provided by the gateway provider which involves a combination of different fields from the payload, some optional fields and most importantly the Salt shared by both parties.
3. After the request reaches the gateway back-end, the back-end calculates the hash of the request payload that is sent using the same formula and matches with the sent hash.
4. A request is only accepted only if both hashes match, otherwise the request is rejected or some error message is returned.

The Processes of the request integrity checking between payment service and payment gateway service is displayed in the figure 4.5. For each response received from the payment gateway into the payment service backed the process is almost similar:

1. With the payload to be sent to the payment service from the gateway as response, the hash calculated from the payload is sent with the response as well.
2. The formula for calculating the hash is done using SHA512 algorithm using a formula provided by the gateway provider which is similar to the previous algorithm but in reverse.
3. After the response reaches the payment service back-end, the back-end calculates the reverse-hash (e.g the same algorithm but in reverse) of the request payload and matches with the hash sent by the gateway.

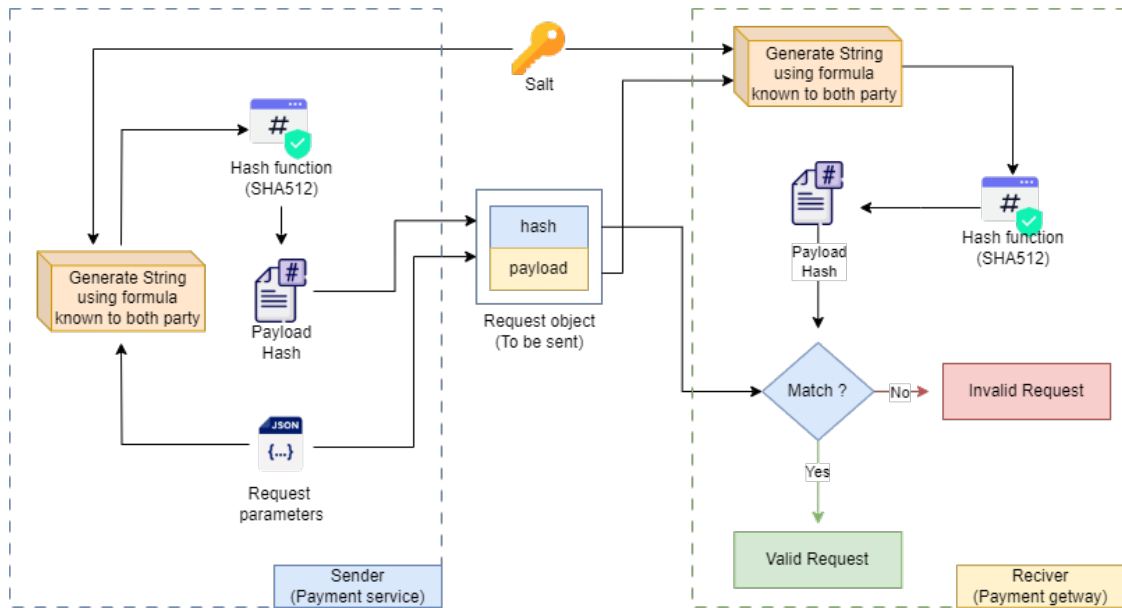


Figure 5.1: Request integrity check between payment service to payment gateway

4. A response is only accepted only if both hashes match, otherwise the response is rejected or some error message is returned.

The Processes of the response integrity checking between payment gateway and payment service is displayed in the figure 4.6.

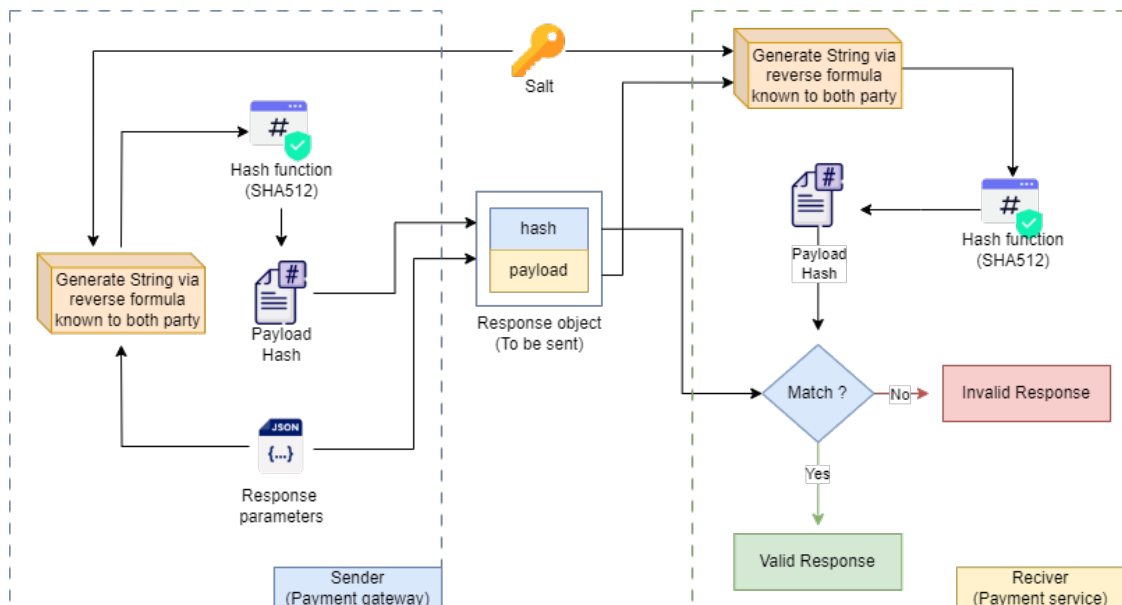


Figure 5.2: Integrity check of response from payment gateway to payment service

After implementing the recommended methods for mitigation and re-developing the affected components that is taking proper checks for integrity of requests and

response, it was observed that the exploitation was unable to be performed any more following the previous process, hence this issue was considered to be closed.

5.2.2 Exposure of critical Information Using Insecure Direct Object Reference (IDOR)

As discussed in the previous sections, Insecure Direct Object Reference or IDOR refers to the a vulnerability in which attackers can access or modify objects by manipulating identifiers in a web application's URLs or parameters. In this case, it was observed that an attacker can use IDOR vulnerability on render time of the main payment screen and get critical information using the order id from the request parameter. Using this vulnerability, the attacker access critical information of different order bypassing access control checks put in place e.g access user information or order information etc, by injecting arbitrary order id in the request parameter. Refer to 4.2.2.1

5.2.2.1 Possible Mitigation Measures

One possible mitigation measure for this type of attack is limiting the access of information from the request URL parameters. So that, any attacker cannot modify the request URL parameters on the way to the destination. To achieve this, the URL parameters of the payment link can be encrypted and during the render time of the page, the URL parameter can be decrypted and from that information necessary data can be fetched and displayed on the payment screen. Therefore, the attacker cannot modify the order id before the render of the screen, and even if the parameters are modified, the result at back-end will be a garbage value during the decryption process, Hence no useful information will be exposed to the attacker.

Also the after decryption of the parameters, the information can be stored in session store of the back-end, hence, if the parameters are modified in future, they can be checked for validity. For exmaple, store the orderId in session of the payment link sent to user, so if the parameters are modified, the value of order id in session can be checked and if both are not equal, then the request is modified on the way and can be rejected.

Another measure that can be taken is to include authorization token for example JSON web token in all requests to protected requests so that only the authorised users can request to the protected resource and enforce access control checks.

5.2.2.2 Mitigation Measures Taken

The measure taken to mitigate this type of vulnerability in this case this discussed below: For this case, the attacker has access to the payment link that contents encoded URL parameter.

1. On hitting the request to the server, the url parameter is decoded and the orderId is stored in the session store of the server.
2. Now all subsequent request that requires the orderId as payload to fetch information (e.g fetch product information, information of the user that placed the order, total amount to be paid), is checked against the orderId stored in session.
3. If both ids match, the proper response is returned otherwise some error response it returned.
4. Hence, after the time of render of the payment page it is ensured that, only the intended orderId is used for getting information.

Following the methods discussed above, the component is re-developed and tested if the vulnerability still exists or not. After test of the component, it was observed that the exploitation was unable to be performed any more following the previous process, hence this issue was considered to be closed. Also other measures (for example usage of access-token for each protected routes) are also explored for this case but due to the scope of the application, the above method was decided to be implemented.

5.2.3 HTML Injection Attack

In this case, the system is found to be vulnerable to HTML injection attacks and the vulnerable components were the success and failure screens. During the assessment and through pen testing of these components it was discovered that attacker can inject malicious code snippet on top of the code of the page at render time of the page. Which can result in execution of malicious code in the system, through this the attacker can steal the session information of a valid user or worse it can be used to plant remote administration tools (RATS) for example Metasploit can be used to perform the same, resulting in the compromise of the application server. The detailed process of the exploitation of this vulnerability is discussed in the previous sections e.g 4.2.3.1

5.2.3.1 Possible Mitigation Measures

One of the most primary measure to mitigate this type of attack is to encode HTML character on the ip/op strings. HTML encoding converts HTML characters (e.g., < and >) into their respective character entities. This helps to prevent HTML injection attacks by treating user-submitted data as plain text rather than HTML code.

It is recommended to used safe APIs that are proven to be resilient to this type of attacks for example For example, `textContent`, `innerText`, methods built in javascript are safe to use. These APIs display information as is, without extracting code. Although These APIs have restrictions for example HTML tags are shown as pure text, preventing their intended interpretation. To disable JavaScript code within untrusted data, the auto-escape approach with escaping modifiers can be used.

Another method that can be implement Content Security Policy (CSP). CSP is a security feature that allows websites to control which sources can load resources including JavaScript, CSS, and pictures. CSP allows websites to prevent the execution of harmful HTML code injected into the page.

Implementing proper Input validation and input sanitation method for each input and output. This can greatly help avoid HTML injection attacks, blocking the submission of harmful HTML code in the payload.

5.2.3.2 Mitigation Measures Taken

For this use case, the affected component is identified that is the success and failure transaction screens. Both the components are checked for the cause of the issue, the cause is identified and proper validation checks are incorporated which encodes the HTML codes in the string into UTF-8 encoded equivalent, hence the code as a whole is not executed which mitigates other other escalated attacks for example XSS.

5.2.4 Insecure Cookie Attribute Configuration

During the initial security assessment it was observed that in the application, the session cookie and access-token-cookie generated by the application, has the HTTP-only and secure as false, this causes the cookies to be exposed by client-side JavaScript and the cookies are transmitted as plain-text as secure attribute is not set. This escalates the risk of session-hijacking attack and potential exposure of critical credential e.g access-token used by the application that is needed for calling

other services such as order processing service.

5.2.4.1 Possible Mitigation Measures

Primary measure that could be taken is to implement HttpOnly and Secure cookie flags for session cookies and other auth-cookies (e.g access-token or JWT cookie) to ensure session IDs are only transmitted over HTTPS channels and they cannot be accessed by client-side script.

5.2.4.2 Mitigation Measures Taken

This is case, the primary measure is taken as explained in section 5.2.4.1 It was ensured that proper cookies settings are put in place in server configurations. Also proper configurations for setting required flags is enable for application functions responsible setting cookies.

5.2.5 CORS Misconfiguration

During the assessment of the service it was observed that the server was able to validate requests from any origin and deliver a successful response due to improper validation of the Origin header. Using this vulnerability an attacker can send a malicious request (i.e get user-data) via a web URL in control of the attacker and capture the response sent by the destination web-server. This way the attacker can access the responses sent by the compromised web-app through a 3rd party malicious Web-application.

5.2.5.1 Possible Mitigation Measures

To mitigate this type of vulnerability it is crucial to make sure that For sensitive web resources, specify the authorized origins explicitly in the Access Control Allow Origin header i.e avoid the use of wild cards. Only trustworthy sites should be allowed in the Access Control Allow Origin header. If the web url is public and does not require authentication, such as an API endpoint, avoid dynamically reflecting cross-domain request headers without validation.

5.2.5.2 Mitigation Measures Taken

In this case, to mitigate this type of attack, proper CORS configurations are put in place at the web-server configuration. Putting wildcard at Access-Control-Allow-Origin is avoided. Only known and trusted origins to the web server are

included in the **Access-Control-Allow-Origin** flag at the service back-end server configuration.

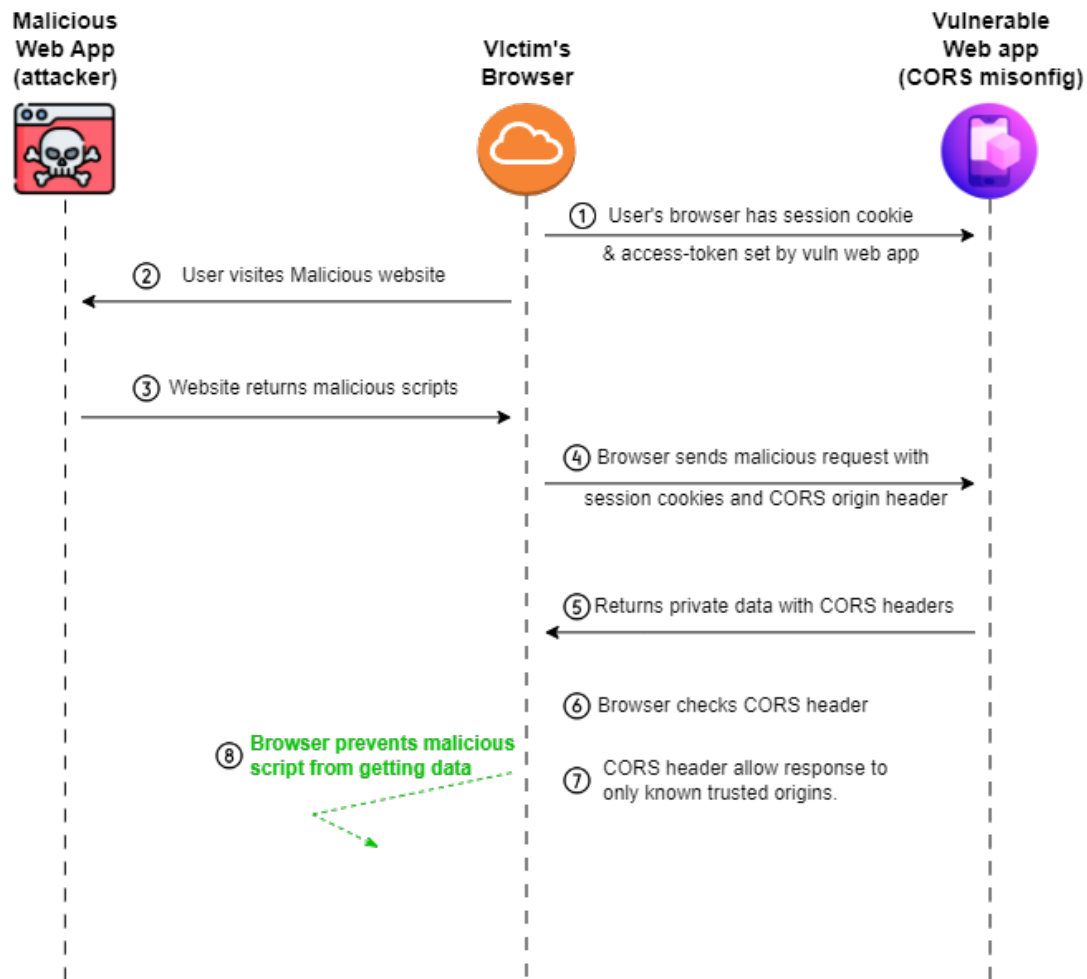


Figure 5.3: Example of CORS configuration preventing external origin attacks [Image courtesy: D. Danelon, The complete guide to cors (in)security]

After adding recommended setting for Cross origin requests and whitelisting trusted domain in the configuration, again tests were conducted by sending request from arbitrary domains, and was observed that responses were rejected by browser throwing CORS error, which was the expected behaviour, hence, how arbitrary origin requests cannot be performed anymore by any attacks. Hence, the issue was decided to be closed.

CHAPTER 6

Conclusions

In this project work, A payment service developed for an enterprise commerce platform following requirements set by relevant stakeholders. Initially, prototype of the system is presented for to demonstrate the basic requirements of the system, following that development, coding and testing is done on top of the prototype to build a working system following SDLC process involving stakeholders. After that security assessment is performed on the payment service. Following the assessment result, discovered vulnerability are discussed, mitigation measures are discussed and finally mitigation measures are taken and vulnerable components are redeveloped following recommended practices. As a result, processing of large online transactions were enabled in the platform, previously online offline transactions was possible. This made, the whole process of ordering products, payments processing ware made easy for the user base. The main aim of this study and development is to better understand enterprise applications especially e-commerce platforms that caters to a large user base and understand best security practices followed for a secure and scalable systems in large enterprises. The results of security assessments and found vulnerabilities are some of the most common web application security vulnerabilities exploited by attackers. therefore, the recommended mitigation measures and best practices discussed in this study can be applied in development of e-commerce platforms or web applications which could be solid foundation for following best practices in the development process. It is to be noted, although extensive studies are done in the field of cyber-security of web applications, and various vulnerabilities of web applications are thoroughly researched, this study builds off of these knowledge base to apply them into a practical development environment and discusses the results of the findings. Hence, this study could act as an addition to the existing knowledge base and be a good reference for development and following good practices of web application and web application security testing.

6.1 Limitations of Current Work

For this project work, the scope of development is only the payment service of the whole e-commerce platform, the same is applied for the security assessment of that is only the payment service was considered as the primary scope for security assessment. Other components and services of the platforms is not considered during in this study although the security of an applications is dependent on not only on one service but the whole platform, for an e-commerce platform a through assessment of the whole platform is necessary because of the scale of the platform, there can be vulnerabilities found in many services throughout the platform.

6.2 Scope for Improvements

In future, this work can be extended to consider a whole e-commerce platform, following development, security assessment can be performed to consider the whole platform. Also in this study, some of the most common security risks are discussed, for example OWASP top 10 security risks. But in future, a more detailed assessment can be performed in order to consider some less common but important security risks for example remote code executions(RCE), crypto-jacking, LFI, RFI, kernel exploits etc.From the comprehensive studies can lead to the development of more generic mitigation techniques and procedures for development of secure platforms.

CHAPTER A

Appendix-I

In this Chapter the the practical overview of the payment system is displayed, using some example requests made to the system and screen-captures are displayed. This chapters also describes all the screens of the payment service for an example payment link, including success and failure pages after success or failure of payments.

A.1 Screens of the Payment Service

The user uses a payment link to get to the main payment page, the main payment screen is displays the order information, total amount, dealer info, etc.

The screenshot shows a web browser window with the URL `dev.payment-serv.com/payment/7f2+wQeExAzwyMl+31lWqrRqOoYnCwc8Yb7HgQRufWpA=`. The page is titled "ORDER DETAILS" and contains the following information:

BOOKING ID : 00102056 | **DATE OF BOOKING :** 2024-04-18 | **DATE OF PO :** NA

CUSTOMER INFO | **DEALER INFO** | **AMC INFO**

Customer Name
Adi Abdal

Email Address
tst.svxc10@gmail.com

Contact Number
7947781579

Customer Address
Canning, Nikarighata, West Bengal 743329

Price Details

Total Amount Rs.171115
*Price includes 18% GST

Product Amount Rs.147125.00

AMC Charges (Extended Rs.23990 Warranty)

Amount to be paid Rs.97552.50
50.00% of Product Amount + 100% of AMC Amount

PAY NOW

ORDER DETAILS

Model Name
Classic

View Full Specifications +

* Representational image. The original product may differ.

* Tentative date & final date may vary post PO placement which will be communicated to you.

* Inclusive of installation charges of INR 2301 (Inclusive of GST) for Doors & INR 17700 (Inclusive of GST) for Windows & ventilator per unit

* Exclusive of payment gateway charges

Figure A.1: Initial payment screen on request to payment-link

On clicking pay now, user is presented with a new screen where the user has the option to pay Online or pay using Offline methods that is pay after delivery. This screen is displayed in the figure below.

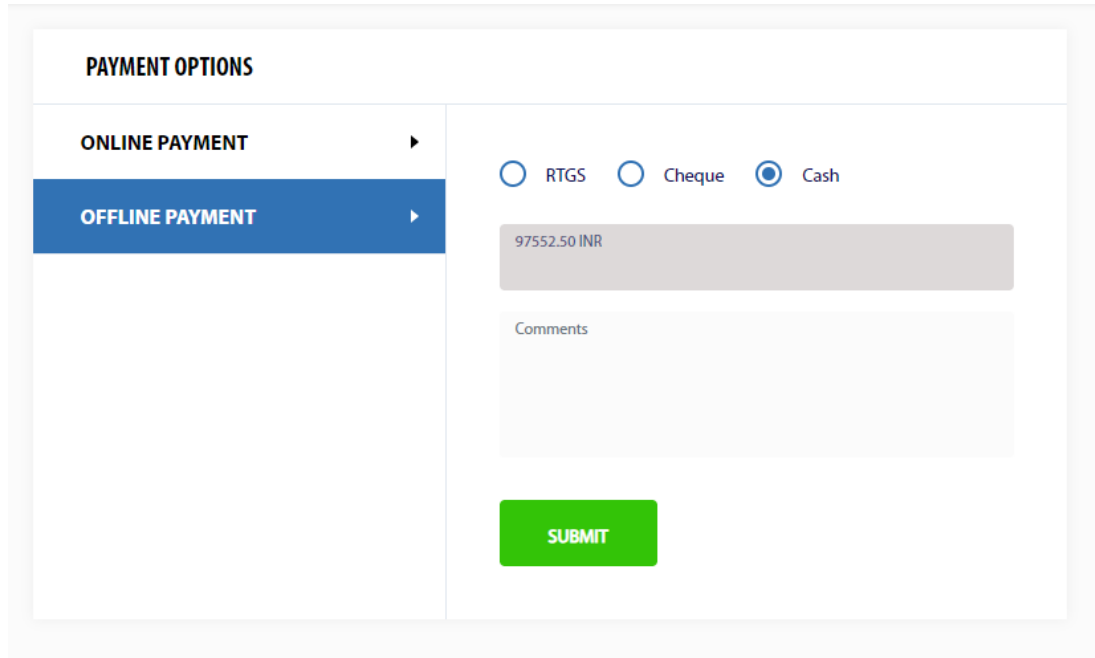


Figure A.2: User redirected to next screen on clicking pay now

On selecting, offline payment, an order is registered against the user and the relevant order-id, the user can complete payment on receiving the ordered products. But selecting Online options, the user is redirected on to the payment gate-way.

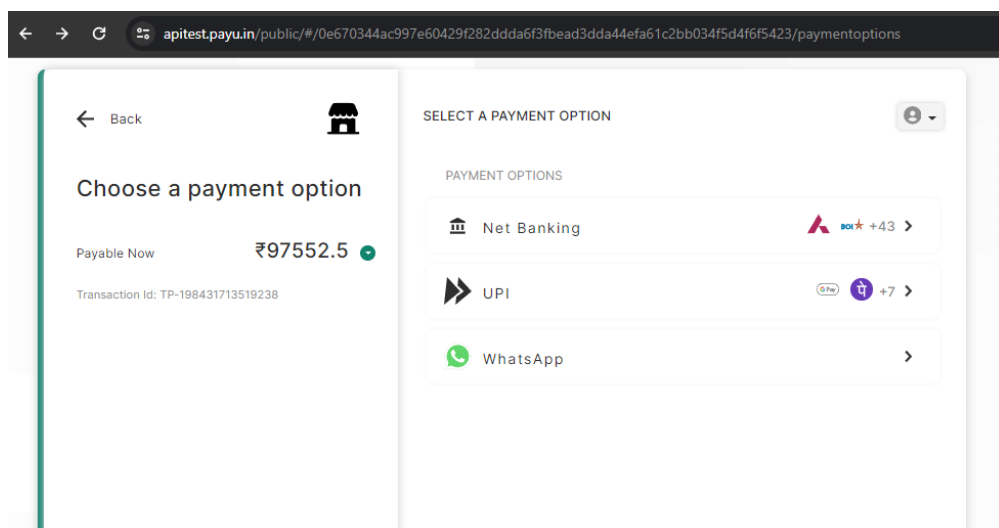


Figure A.3: Payment gateway for online payment

On completions of a transaction, the response sent by the gate-way is sent to the

payment service, where the response is processed and the user is redirected to the success or failure screen based on the transactions response. E.g for a failed payment, the user is redirected to failure screen.

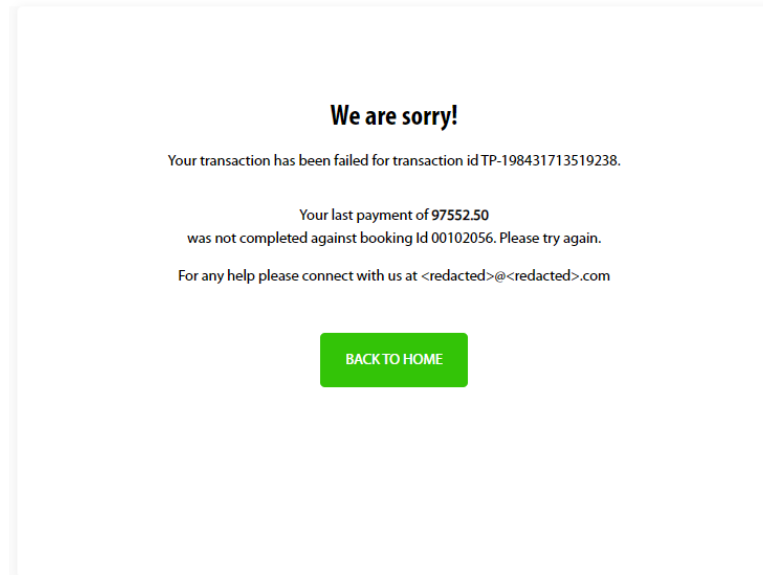


Figure A.4: Fail screen on a failed transaction

For a successful payment, the user is redirected to the success screen as described in the figure A.5.

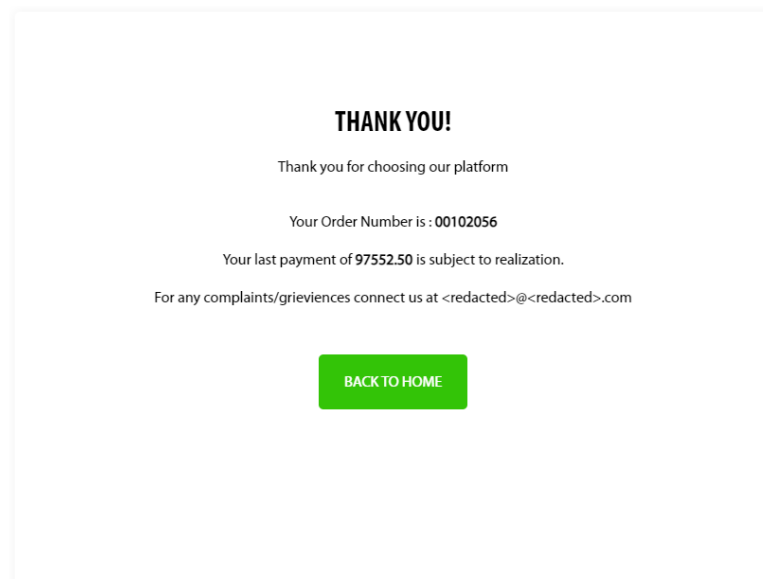


Figure A.5: Success screen on a successful transaction

A.2 Sample Code Snippets

Below are some code snippets used in development of different components in the payment system of the e-commerce platform.

```

1  <?php
2  include_once("aes_enc_dec.php");
3
4  $key="";
5  $salt="";
6  //$action = 'https://test.payu.in/_payment';
7  $action='https://secure.payu.in/_payment';
8  $dealerId = !empty($returnArr['dealerId']) ? $returnArr['dealerId'] : '';
9  $payDetArr = constant("PAYU_DETAILS");
10 if(!empty($dealerId)) {
11     $key = isset($payDetArr[$dealerId]['key']) ? $payDetArr[$dealerId]['key'] : '';
12     $salt = isset($payDetArr[$dealerId]['salt']) ? $payDetArr[$dealerId]['salt'] : '';
13 }
14 // encode salt using aes encryption to be used at success and fail page
15 $encoded_salt = encrypt_payload($salt);
16 if(!isset($_SESSION)) session_start();
17 $_SESSION["merchant_data_enc"] = $encoded_salt;
18
19 $bookingId = isset($returnArr['bookingId']) ? $returnArr['bookingId'] : (isset($_SESSION['bookingId']) ?
20 $firstName = isset($returnArr['bookingName']) ? $returnArr['bookingName'] : '';
21 $lastName = '';
22 $phoneNo = isset($returnArr['customerContactNo']) ? $returnArr['customerContactNo'] : '';
23 $email = isset($returnArr['customerEmail']) ? $returnArr['customerEmail'] : '';
24 $productName = 'Test product';
25 $city = isset($returnArr['bookingStreet']) ? $returnArr['bookingStreet'] : '';
26 $state = isset($returnArr['bookingState']) ? $returnArr['bookingState'] : '';
27 $country = isset($returnArr['bookingCountry']) ? $returnArr['bookingCountry'] : '';
28 $district = isset($returnArr['bookingDistrict']) ? $returnArr['bookingDistrict'] : '';
29 $postcode = isset($returnArr['bookingZipcode']) ? $returnArr['bookingZipcode'] : '';
30 $address1 = $city.', '.$district.'.';
31 $address2 = $state.', '.$postcode.'.';
32 $udf1 = isset($_SESSION['url_parameter']) ? $_SESSION['url_parameter'] : '';
33 $udf2 = isset($_SESSION['updatedCPDDDate']) ? $_SESSION['updatedCPDDDate'] : '';
34 $udf3 = isset($_SESSION['updatedCPDDReason']) ? $_SESSION['updatedCPDDReason'] : '';
35
36 $html= '' ;
37 // $amount = 10;
38 $amount = isset($amount) ? $amount : (isset($returnArr['bookingAmount']) ? $returnArr['bookingAmount'] :
39 $txnId = 'TP-'.substr(str_shuffle("0123456789"), 0, 5).'.'.time();
40 $amount = str_replace('₹', '', $amount);
41 $amount = str_replace(',', '', $amount);
42 //generate hash with mandatory parameters and udf5
43 $hash=hash('sha512', $key.'|'.$txnId.'|'.$amount.'|'.$productName.'|'.$firstName.'|'.$email.'|'.$udf1.'|'
44
45 // $success_data="bookingId=$bookingId&payMode=$payMode";
46 // $encrypt_success_data=encrypt($success_data);
47 $successUrl = DOMAIN_URL.DOMAIN_PATH."success.php?bookingId=$bookingId&payMode=$payMode";
48 $failUrl = DOMAIN_URL.DOMAIN_PATH."fail.php?bookingId=$bookingId";

```

Figure A.6: Code snippet of payment gateway integration

Figure A.6 presents a code snippet for integration of payment gateway (PayU) into the e-commerce platform. The service has key and salt that are decided between the service and the gateway, it fetches order details from other endpoints of the service and use this information to construct the payload to be sent to the gateway with hash calculated using key, salt and other information. The hash is calculated using the order data i.e order payload along with the key and salt using an algorithm

decided by both parties which is in this case the payment service and the payment gateway provider.

```

1  <?php
2  /**
3   * this function calculates the hash (reversed) with data received
4   * by $_POST response sent by payU and matches with hash sent by payU.
5   * return true if match, else false
6   * @param $RESPONSE=$_POST
7   * @param $SALT=string
8   * @return bool
9   */
10 function validate_response_hash($RESPONSE, $SALT) {
11     // return is empty response data
12     if (empty($RESPONSE)) {
13         return false;
14     }
15
16     $firstname    = isset($RESPONSE["firstname"]) ? $RESPONSE["firstname"] : '';
17     $status       = isset($RESPONSE["status"]) ? $RESPONSE["status"] : '';
18     $amount       = isset($RESPONSE["amount"]) ? $RESPONSE["amount"] : '';
19     $txnid        = isset($RESPONSE["txnid"]) ? $RESPONSE["txnid"] : '';
20     $response_hash = isset($RESPONSE["hash"]) ? $RESPONSE["hash"] : '';
21     $key          = isset($RESPONSE["key"]) ? $RESPONSE["key"] : '';
22     $productinfo  = isset($RESPONSE["productinfo"]) ? $RESPONSE["productinfo"] : '';
23     $email        = isset($RESPONSE["email"]) ? $RESPONSE["email"] : '';
24     //udf5
25     $udf5         = isset($RESPONSE["udf5"]) ? $RESPONSE["udf5"] : '';
26     $udf1         = isset($RESPONSE["udf1"]) ? $RESPONSE["udf1"] : '';
27     $udf2         = isset($RESPONSE["udf2"]) ? $RESPONSE["udf2"] : '';
28     $udf3         = isset($RESPONSE["udf3"]) ? $RESPONSE["udf3"] : '';
29
30
31     $calculated_string = '';
32     $calculated_hash   = '';
33
34     if (isset($_POST["additionalCharges"])) {
35         $additionalCharges = $RESPONSE["additionalCharges"];
36         $calculated_string = $additionalCharges.'|'.$SALT.'|'.$status.'|'.$udf5.'|'.$udf3.'|'.$udf2.'|'.
37     } else {
38         $calculated_string = $SALT.'|'.$status.'|'.$udf5.'|'.$udf3.'|'.$udf2.'|'.$udf1.'|'.$email.'|'.
39     }
40
41     $calculated_hash = hash('sha512', $calculated_string);
42
43     if ($calculated_hash != $response_hash) {
44         return false;
45     }
46     return true;
47 }
48 ?>

```

Figure A.7: Code snippet of integrity check of response from payment gateway (Client side)

Figure A.7 presents a code snippet for response integrity check function that checks the responses that comes from the payment gateway to the client i.e the the e-commerce platform in this case. it checks the hash sent with the response with the hash calculated with response payload and the salt that is a secret between the client and the payment gateway service.

```

1  <?php
2  date_default_timezone_set('Asia/Calcutta');
3  include_once('config.php');
4  if(!isset($_SESSION)) session_start();
5
6  $session_booking_id = isset($_SESSION['bookingid']) ? $_SESSION['bookingid'] : '';
7
8  $fromPage = isset($_POST['from']) ? $_POST['from'] : 0;
9  $payFrom = isset($_POST['additionalPayment']) ? $_POST['additionalPayment'] : ( isset($_GET['additionalPa
10 $expiry = 1800 ; //cookies expiry required after 30 mins
11 $cBid = (!empty($_SESSION['bookingId'])) ? $_SESSION['bookingId'] : '';
12 $bookingId = (!empty($_POST['bookingId'])) ? $_POST['bookingId'] : $cBid;
13
14 if(intval($fromPage) < 4) {
15     if ($bookingId != $session_booking_id) {
16         // header('HTTP/1.1 400 Bad Request');
17         echo 'Bad Request -- invalid booking id';
18         exit();
19     }
20 }
21
22 if(!isset($_SESSION['bookingId'])) {
23     $_SESSION['bookingId'] = null;
24 }
25 if(!empty($bookingId) && $bookingId != $cBid){
26     $_SESSION['bookingId']=$bookingId;
27 }
28
29 $authorization = isset($_COOKIE['token']) ? $_COOKIE['token'] : '';
30 $response = '';
31 //Access Token
32 if(empty($authorization)) {
33     $curl = curl_init();
34     curl_setopt_array($curl, array(
35         CURLOPT_URL => AUTH_API,
36         CURLOPT_RETURNTRANSFER => true,
37         CURLOPT_ENCODING => '',
38         CURLOPT_MAXREDIRS => 10,
39         CURLOPT_TIMEOUT => 0,
40         CURLOPT_FOLLOWLOCATION => true,
41         CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
42         CURLOPT_CUSTOMREQUEST => 'POST',
43         CURLOPT_POSTFIELDS => array('grant_type' => 'password','client_id' => CLIENT_ID,'client_secret' =>
44         CURLOPT_HTTPHEADER => array(
45             'Cookie: BrowserId=TabSFQatEey9-2WMU2q1kg; CookieConsentPolicy=0:0'
46         ),
47     ));
48     $response = curl_exec($curl);

```

Figure A.8: Code snippet of getting order data using orderId and access token

Figure A.8 presents a code snippet this is used in the payment service to fetch order details using the order id fetched from the request parameters At initial payment page, total payable amount is displayed from all the orders placed, this information is fetched from the order ID through a part of the back-end code provided in this snippet. All requests to the cart service are to be made an access token for security as it is a critical endpoint.

```

2  <html lang="en">
22 <body>

44     <?php
45
46     /**
47      * Guard clause 1
48      * check for invalid render
49      * i.e Page is rendered by other parties (as a GET request) rather than
50      * as a response to POST request sent by payU in response to a payment transaction
51      */
52
53     if (empty($_POST)) {
54         header('HTTP/1.1 400 Bad Request');
55         echo 'Bad Request -- invalid render';
56         exit();
57     }
58
59     // Validate the presence of critical required parameters
60     if (!isset($_POST['status']) || !isset($_POST['hash'])) {
61         header('HTTP/1.1 400 Bad Request');
62         echo 'Bad Request -- invalid response';
63         exit();
64     }
65
66     ?>
67
68     <?php
69     $txnId = isset($_POST['txnId']) ? $_POST['txnId'] : '';
70     $cTxnid = (!empty($_COOKIE['txnId'])) ? trim($_COOKIE['txnId']) : '';
71
72     $bookingId = !empty($_GET['bookingId']) ? $_GET['bookingId'] : (!empty($_SESSION['bookingId']) ? $_
73
74     $is_valid_signature = validate_response_hash($_POST, $salt); // check response signature
75     // $txnId !== $cTxnid
76     if ($is_valid_signature == true && $_POST["status"] == 'success') {
77         $jsonVal = json_encode($_POST); ?>
78         <input type="hidden" id="payVal" value='<?php echo $jsonVal;?>'>
79         <div class="main-container">
80             <?php $show = 'show';include_once('thankyou.php'); ?>
81         </div>
82         <!-- updated-jquery -->
83         <!-- <script src="assets/vendors/js/jquery-3.6.0.min.js"></script> -->
84         <script src="assets/vendors/js/jquery-3.7.0.min.js"></script>
85
86         <script src="assets/vendors/js/bootstrap.bundle.min.js"></script>
87         <!-- <script src="https://code.jquery.com/ui/1.13.1/jquery-ui.js"></script> -->
88         <script src="https://code.jquery.com/ui/1.13.2/jquery-ui.js" integrity="sha256-xLD7nhI62fcsEZK;

```

Figure A.9: Code snippet of success screen along with response validity check

In Figure A.9 a code snippet is presented which is used in the success screen component. It takes the response from the payment gateway as input, depending on the response, it renders the success screen. Hence, the response integrity check is required to mitigate any tampering attack between the gateway response and the payment service. In this snippet and previous snippet A.7 the response validity check is presented.

```

2   <html lang="en">
22  <body>
58  ?>
59
60  <?php
61  $txnId = isset($_POST['txnId']) ? $_POST['txnId'] : '';
62  $cTxnid = (empty($_COOKIE['txnId'])) ? trim($_COOKIE['txnId']) : '';
63  $bookingId = !empty($_GET['bookingId']) ? $_GET['bookingId'] : (empty($_COOKIE['bookingId']) ? $_COO
64  $arr = array();
65
66
67  $decryptedData = isset($_SESSION['booking_data']) ? decrypt_data_of_booking($_SESSION['booking_data'])
68  $returnArr = isset($_SESSION['booking_data']) ? json_decode($decryptedData, 1) : '';
69  //var_dump($re0PQoNwBA0Y6FL6wFW );
70
71  $dataArr = !empty($_POST) ? $_POST : array();
72  $arr['amount'] = isset($dataArr['amount']) ? $dataArr['amount'] : '';
73  $arr['paymentType'] = 'Online';
74  $arr['paymentMode'] = isset($dataArr['mode']) ? $dataArr['mode'] : 'payU';
75  $arr['paymentStatus'] = isset($dataArr['status']) ? $dataArr['status'] : '';
76  $arr['transactionId'] = isset($dataArr['txnId']) ? $dataArr['txnId'] : '';
77  $arr['bankRefNo'] = isset($dataArr['bank_ref_num']) ? $dataArr['bank_ref_num'] : '';
78  $arr['paymentDateTime'] = isset($dataArr['addedon']) ? $dataArr['addedon'] : '';
79  $arr['paymentRemarks'] = !empty($dataArr['error_Message']) ? $dataArr['error_Message'] : '';
80  $jsonVal = json_encode($arr);
81
82  $is_valid_signature = validate_response_hash($_POST, $salt);
83
84  // if ($txnId != $cTxnid && !empty($txnId)) {
85  if ($is_valid_signature == true && $_POST["status"] == 'failure') {?>
86  <input type="hidden" id="payVal" value="<?php echo $jsonVal; ?>">
87  <div class="main-container">
88  <div class="fail-pop thank-pop" id="failpop">
89  <div class="modal-dialog modal-lg modal-dialog-centered">
90  <div class="modal-content">
91  <div class="thank-you">
92  <h1>We are sorry!</h1>
93  <p>Your transaction has been failed for transaction id <?php echo $txnId; ?>.
94  </div>
95  <div class="last-payment">
96  <p class="trid">Your last payment of <strong id='amt_val'><?php echo $arr['am
97  <p class="your-order-text">For any help please connect with us at helpline.pr
98  </div>
99  <?php
100  if(isset($_COOKIE['url_parameter'])) {
101  $enc_data = $_COOKIE['url_parameter'];
102  //echo "Value of the cookie 'cookie_name' is: " . $_COOKIE['url_parameter'];
103  } else {
104  //echo "Cookie 'cookie_name' is not set ";

```

Figure A.10: Code snippet of failure screen along with response validity check

In Figure A.10 a code snippet is presented which is used in the failure screen component. It takes the response from the payment gateway as input, depending on the response, it renders the failure screen.

Bibliography

- [1] V. van Gelder. “E-commerce worldwide - statistics and facts.” (2024), [Online]. Available: <https://www.statista.com/topics/871/online-shopping/>.
- [2] Wikipedia, *Symmetric key algorithm*, https://en.wikipedia.org/wiki/Symmetric-key_algorithm, [Online; accessed 09-April-2024], 2024.
- [3] H. Nahari and R. L. Krutz, *Web commerce security: design and development*. John Wiley & Sons, 2011.
- [4] Wikipedia, *Botnet Wikipedia, the free encyclopedia*, <http://en.wikipedia.org/w/index.php?title=Botnet&oldid=1200010008>, [Online; accessed 09-April-2024], 2024.
- [5] “Sql injection.” (2024), [Online]. Available: https://owasp.org/wwwcommunity/attacks/SQL_Injection.
- [6] “Owasp top ten.” (2024), [Online]. Available: <https://owasp.org/www-project-top-ten/>.
- [7] Wikipedia, *Gray-box testing — Wikipedia, the free encyclopedia*, <http://en.wikipedia.org/w/index.php?title=Gray-box%20testing&oldid=1196060044>, [Online; accessed 07-April-2024], 2024.
- [8] L. J. Trautman, “E-commerce, cyber, and electronic payment system risks: Lessons from paypal,” *UC Davis Bus. LJ*, vol. 16, p. 261, 2015.
- [9] S.-H. Chun, “E-commerce liability and security breaches in mobile payment for e-business sustainability,” *Sustainability*, vol. 11, no. 3, p. 715, 2019.
- [10] C. K. N. Jerrin Yomas, “A critical analysis on the evolution in the e-payment system, security risk, threats and vulnerability,” *Communications on Applied Electronics*, vol. 7, no. 23, pp. 21–29, Dec. 2018, ISSN: 2394-4714. DOI: 10.5120/cae2018652800.

-
- [11] M. Niranjana Murthy and D. Chahar, “The study of e-commerce security issues and solutions,” *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 7, pp. 2885–2895, 2013.
 - [12] J.-H. Yang, Y.-F. Chang, and Y.-H. Chen, “An efficient authenticated encryption scheme based on ecc and its application for electronic payment,” *Information Technology And Control*, vol. 42, Dec. 2013. DOI: 10.5755/j01.itc.42.4.2150.
 - [13] S. Chaudhry, M. Farash, S. Naqvi, and M. S. Ramzan, “A secure and efficient authenticated encryption for electronic payment systems using elliptic curve cryptography,” *Electronic Commerce Research*, vol. 16, Jun. 2015. DOI: 10.1007/s10660-015-9192-5.
 - [14] F. Miranda González and T. Bañegil Palacios, “Quantitative evaluation of commercial web sites: An empirical study of spanish firms,” *International Journal of Information Management*, vol. 24, no. 4, pp. 313–328, 2004, ISSN: 0268-4012. DOI: <https://doi.org/10.1016/j.ijinfomgt.2004.04.009>.
 - [15] S. Solat, *Security of electronic payment systems: A comprehensive survey*, 2017.
 - [16] S. Nambiar, C.-T. Lu, and L. Liang, “Analysis of payment transaction security in mobile commerce,” in *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, 2004. IRI 2004.*, 2004, pp. 475–480. DOI: 10.1109/IRI.2004.1431506.
 - [17] S. MKTG1, *10 Types of Security Testing Techniques | SISA Insights — sisainfosec.com*, <https://www.sisainfosec.com/blogs/10-types-of-security-testing-techniques/>, [Accessed 10-04-2024].
 - [18] *Penetration test - Wikipedia — en.wikipedia.org*, https://en.wikipedia.org/wiki/Penetration_test, [Accessed 10-04-2024].
 - [19] *Nessus — cs.cmu.edu*, <https://www.cs.cmu.edu/~dwendlan/personal/nessus.html>, [Accessed 10-04-2024].
 - [20] *Public-key cryptography*, https://en.wikipedia.org/wiki/Public-key_cryptography, [Accessed 10-04-2024].
 - [21] *Symmetric-key algorithm*, https://en.wikipedia.org/wiki/Symmetric-key_algorithm, [Accessed 10-04-2024].

- [22] C. Stohrer and T. Lugrin, “Asymmetric encryption,” in *Trends in Data Protection and Encryption Technologies*, V. Mulder, A. Mermoud, V. Lenders, and B. Tellenbach, Eds. Cham: Springer Nature Switzerland, 2023, pp. 11–14, ISBN: 978-3-031-33386-6. DOI: 10.1007/978-3-031-33386-6_3.
- [23] N. Krzyworzeka, “Asymmetric cryptography and trapdoor one-way functions,” *Automatyka/Automatics*, vol. 20, no. 2, pp. 39–51, 2016.
- [24] Q. Dang, *Secure hash standard*, en, 2015. DOI: <https://doi.org/10.6028/NIST.FIPS.180-4>.
- [25] Wikipedia, *JQuery — Wikipedia, the free encyclopedia*, <http://en.wikipedia.org/w/index.php?title=jQuery&oldid=1217841529>, [Online; accessed 12-April-2024], 2024.
- [26] Wikipedia, *PHP — Wikipedia, the free encyclopedia*, <http://en.wikipedia.org/w/index.php?title=PHP&oldid=1218465312>, [Online; accessed 12-April-2024], 2024.
- [27] *Payu.in*, <https://payu.in/>, [Accessed 12-04-2024].
- [28] *Insecure Direct Object Reference Prevention - OWASP Cheat Sheet Series — cheatsheetseries.owasp.org*, https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html, [Accessed 16-04-2024].
- [29] *WSTG - Latest | OWASP Foundation — owasp.org*, https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection, [Accessed 16-04-2024].
- [30] *Cross-origin resource sharing - Wikipedia — en.wikipedia.org*, https://en.wikipedia.org/wiki/Cross-origin_resource_sharing, [Accessed 17-04-2024].
- [31] *WSTG - Latest | OWASP Foundation — owasp.org*, https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/07-Testing_Cross-Origin_Resource_Sharing, [Accessed 02-05-2024].
- [32] *Cross-site request forgery - Wikipedia — en.wikipedia.org*, https://en.wikipedia.org/wiki/Cross-site_request_forgery, [Accessed 17-04-2024].