



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Masters Project

ACFA7003

Student Name : Jay Shaileshbhai Lad

Student ID:2504388

Program: MSc Cybersecurity and Digital Forensics

Supervision To: Dr. Fahad Ahmed

ABSTRACT

Web applications are integral to modern organisational operations but remain exposed to a continually evolving range of cyber threats, including SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) and business-logic abuse. This project evaluates and enhances the security of a locally hosted Loyalty Card Management System through a full Vulnerability Assessment and Penetration Testing (VAPT) lifecycle that extends beyond assessment into verified remediation. Using a pragmatic, quantitative experimental research design, the study establishes a controlled, isolated testing environment using Kali Linux and a XAMPP server, guided throughout by the OWASP Top Ten framework. A baseline assessment combining automated scanning (OWASP ZAP, Nikto, SQLMap) with manual penetration testing (Burp Suite Community Edition) identified that, while the application correctly resisted SQL Injection and XSS, it contained critical Cross-Site Request Forgery weaknesses in sensitive state-changing operations, together with missing HTTP security headers and unhardened session cookies. Beyond this baseline, the project makes two further contributions that extend the work to Master's level. First, every identified weakness was remediated at source-code level - anti-CSRF tokens, hardened session cookies, a full security-header policy, brute-force lockout, verified password-reset delivery, email verification and a lightweight CAPTCHA were implemented and are documented with the exact code-level changes applied, ready for empirical re-verification. Second, the study introduces a novel testing dimension largely absent from generic OWASP-based assessments: business-logic vulnerabilities specific to loyalty-reward platforms, namely price tampering, an Insecure Direct Object Reference in order cancellation, and a race-condition double-spending flaw in points redemption. Each was analysed, demonstrated conceptually against the original implementation, and mitigated through server-side price authority, ownership-scoped queries and atomic check-and-set database transactions respectively. The project demonstrates that combining automated scanning, manual verification, source-level remediation and domain-specific business-logic analysis provides a substantially more complete security assurance process than vulnerability scanning alone, and offers a reusable template for assessing loyalty-based e-commerce systems.

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my project supervisor for their invaluable guidance, continuous support, and constructive feedback throughout the development of this research project. Their technical expertise and academic encouragement were instrumental in helping me successfully navigate the complexities of web application security and penetration testing. I also extend my sincere thanks to the faculty and staff at the University of Wales Trinity Saint David for providing a highly supportive academic environment and the essential resources required to complete my MSc in Computer Networks and Cyber Security. Furthermore, I am deeply grateful to my family and friends for their unwavering emotional support, immense patience, and understanding during the long hours of rigorous practical testing and documentation. Finally, I would like to thank my academic peers for their insightful discussions. This project would not have been successfully completed without this collective support.

TABLE OF CONTENTS

ABSTRACT.....	2
ACKNOWLEDGEMENT	3
TABLE OF CONTENTS.....	4
CHAPTER 1: INTRODUCTION	8
1.1 Introduction.....	8
1.2 Background of the Study	8
1.3 Problem Statement.....	9
1.4 Research Aim and Objectives	9
1.5 Research Questions	10
1.6 Scope of the Research.....	10
1.7 Dissertation Structure.....	10
CHAPTER 2: LITERATURE REVIEW	11
2.1 Introduction.....	11
2.2 Web Application Security and Common Threats	11
2.3 OWASP Top Ten as a Security Framework	11
2.4 Vulnerability Assessment and Penetration Testing (VAPT) Methodologies.....	12
2.5 Security Testing Tools	12
2.6 Security Controls and Remediation Techniques	13
2.7 Business-Logic Vulnerabilities in Web Applications	13
2.8 Conceptual Framework.....	14
2.9 Research Gap	14
2.10 Summary.....	15
CHAPTER 3: RESEARCH METHODOLOGY	16
3.1 Introduction.....	16
3.2 Research Approach	16
3.3 Research Design.....	16
3.4 Tools and Techniques	16
3.5 Experimental Setup.....	17
3.6 Ethical Considerations	17
3.7 Summary	17
CHAPTER 4: DESIGN PROCESS	18
4.1 Introduction.....	18
4.2 System Overview and Features.....	18
4.3 Experimental Environment	18
4.4 Application Interfaces	19
4.4A Design Rationale	23
4.5 Pre-Assessment Functional Verification.....	24

4.6 Summary	24
5.1 Introduction.....	25
5.2 Scanning and Reconnaissance	25
5.3 Authentication Security Testing.....	27
5.4 SQL Injection Testing.....	29
5.5 Cross-Site Scripting (XSS) Testing	32
5.6 Cross-Site Request Forgery (CSRF) Testing	33
5.6.1 Profile Modification.....	33
5.6.2 Password Modification	35
5.6.3 Administrative Account Deletion	36
5.7 Security Headers and Session Cookie Analysis.....	37
5.8 Input Validation and Session Management Testing.....	38
5.9 Brute-Force Login Testing.....	39
5.10 Summary of Baseline Findings	40
CHAPTER 6: EVALUATION OF BASELINE RESULTS.....	41
6.1 Introduction.....	41
6.2 Summary of Baseline Security Testing Results	41
6.3 Evaluation of Identified Vulnerabilities.....	41
6.4 Evaluation of Effective Security Controls	42
6.4A Discussion: Why the Effective Controls Matter	43
6.5 Comparison with Research Objectives	43
6.6 Limitations of the Baseline Assessment	43
6.7 Chapter Summary	44
CHAPTER 7: REMEDIATION AND RE-TESTING	45
7.1 Introduction.....	45
7.2 Remediation Methodology.....	45
7.3 CSRF Remediation	45
Verification	46
7.4 Credential and Configuration Hardening	48
Verification	48
7.5 Authentication Hardening	48
7.5.1 Brute-Force Lockout.....	48
7.5.2 Password-Reset Delivery	49
7.5.3 Email Verification.....	49
7.5.4 Bot Protection	49
Verification	49
7.6 Password Policy	50
7.7 Security Headers and Session Cookie Hardening	50
Verification	51

7.8 Re-Testing Methodology	51
7.8A Rationale for Shared, Reusable Security Modules.....	52
7.9 Summary of Remediation Outcomes	52
7.10 Chapter Summary	53
CHAPTER 8: BUSINESS-LOGIC VULNERABILITIES - A NOVEL CONTRIBUTION	54
8.1 Introduction.....	54
8.2 Rationale for a Business-Logic Testing Layer.....	54
8.3 Price Tampering in Order Placement.....	54
8.3.1 Vulnerability Description.....	54
8.3.2 Testing Methodology	54
8.3.3 Mitigation Implemented.....	55
8.4 Insecure Direct Object Reference in Order Cancellation.....	55
8.4.1 Vulnerability Description.....	55
8.4.2 Testing Methodology	55
8.4.3 Mitigation Implemented.....	55
8.5 Double-Spending of Points via Race Condition	55
8.5.1 Vulnerability Description.....	55
8.5.2 Testing Methodology	56
8.5.3 Mitigation Implemented.....	56
8.6 Supporting Control: Tamper-Evident Audit Logging.....	56
8.6A Why These Three Vulnerabilities Were Selected	56
8.7 Evaluation Methodology.....	57
8.8 Chapter Summary	57
CHAPTER 9: CONCLUSIONS AND RECOMMENDATIONS	59
9.1 Conclusion	59
9.2 Recommendations.....	59
CHAPTER 10: REFLECTION.....	61
REFERENCES	62
APPENDIX A: KEY SOURCE-CODE LISTINGS	64
A.1 CSRF Token Handling (includes/csrf.php).....	64
A.2 Secure Session Bootstrap (includes/secure-session.php)	64
A.3 Password Complexity Policy (includes/password-policy.php).....	64
A.4 Lightweight CAPTCHA (includes/captcha.php)	65
APPENDIX B: SUPPLEMENTARY TESTING EVIDENCE	66
B.1 Full Post-Remediation ZAP Alert Summary.....	66
B.2 Rate-Limiting Table Evidence	66

CHAPTER 1: INTRODUCTION

1.1 Introduction

This chapter introduces the project, situating it within the broader context of web application security. It presents the background to the study, defines the problem being addressed, states the research aim and objectives, poses the guiding research questions, sets out the scope of the work, and outlines the structure of the remaining chapters.

1.2 Background of the Study

Web applications now underpin a large proportion of customer-facing business functionality, including loyalty and rewards programmes that store personally identifiable information, transaction history and points balances that carry real monetary value. Attackers increasingly target this class of application because the data held is both sensitive and directly monetisable through fraud, account takeover or resale [1]. Despite widespread awareness of frameworks such as the OWASP Top Ten, many student-built and small-organisation web applications continue to exhibit foundational weaknesses - missing request-forgery protection, absent security headers, unhardened session cookies and hard-coded database credentials - because security is often treated as an afterthought to functional development rather than as an integral part of the software development lifecycle [2].

Vulnerability Assessment and Penetration Testing (VAPT) provides a structured methodology for uncovering such weaknesses before they are exploited in production [3]. However, a VAPT exercise that stops at identification, without progressing to verified remediation, leaves an organisation with a list of known risks rather than a more secure system. This project therefore treats VAPT as a full lifecycle - assess, remediate, and re-verify - applied to a purpose-built Loyalty Card Management System developed specifically for this study.

A further gap motivating this project is that standard VAPT methodologies, and the OWASP Top Ten in particular, are oriented around generic technical vulnerability classes and give comparatively little attention to business-logic flaws that are specific to a given application domain [4]. For a loyalty-reward platform, business logic itself - how points are earned, redeemed and cancelled - represents a distinct attack surface that a purely OWASP-driven assessment can overlook. This project accordingly extends its scope to include domain-specific business-logic testing alongside conventional technical testing.

Loyalty and rewards platforms present a particularly instructive case for studying this problem because they combine several categories of sensitive functionality within a single, comparatively small application: customer authentication and personal data, a stored-value points economy, an order-and-fulfilment workflow, and a privileged administrative interface capable of adjusting customer balances directly. This concentration of functionality means that a single application can meaningfully illustrate both conventional, OWASP-aligned technical vulnerabilities and the kind of business-logic vulnerabilities that are increasingly recognised as a distinct and under-tested risk category [4]. It also means that the consequences of an unremediated vulnerability are concrete and easy to reason about: a CSRF vulnerability in administrative account deletion can remove legitimate administrative access entirely, and a double-spending flaw in points redemption can be directly quantified in terms of the value fraudulently obtained.

1.3 Problem Statement

Student- and small-team-developed web applications frequently combine functionally complete features with incomplete security controls, because development effort is naturally prioritised toward delivering working functionality. The Loyalty Card Management System used in this study exhibited exactly this pattern during its initial build: authentication, a customer dashboard, a shop and order workflow, and an administrative panel were all functionally complete, yet an initial security review identified that sensitive state-changing actions - profile updates, password changes and administrative account deletion - could be performed without any Cross-Site Request Forgery protection, that recommended HTTP security headers were entirely absent, that session cookies lacked hardening attributes, and that database credentials were hard-coded with a blank password for the MySQL root account. In addition, because the application implements a loyalty-points economy, it is exposed to business-logic risks - price tampering, unauthorised order cancellation and double-spending of points - that fall outside the vulnerability classes a generic OWASP-based assessment is designed to catch. The problem this project addresses is therefore twofold: first, to systematically identify and remediate the conventional technical vulnerabilities present in the system, and second, to identify and mitigate the domain-specific business-logic vulnerabilities that a conventional assessment would not surface.

1.4 Research Aim and Objectives

The aim of this project is to assess, remediate and re-verify the security of a locally hosted Loyalty Card Management System through an applied VAPT methodology that combines OWASP-aligned technical testing with domain-specific business-logic analysis.

The objectives are to:

- Design and implement a functionally representative Loyalty Card Management System (customer registration, authentication, dashboard, shop, orders, rewards redemption, and an administrative panel).
- Conduct a baseline VAPT assessment against the OWASP Top Ten using automated tools (OWASP ZAP, Nikto, SQLMap) supported by manual verification with Burp Suite Community Edition.
- Remediate every confirmed vulnerability at source-code level and document the precise changes applied.
- Identify and mitigate business-logic vulnerabilities specific to a loyalty-reward platform, namely price tampering, order-cancellation IDOR, and points double-spending.
- Establish a re-testing methodology to verify that remediated controls function as intended.
- Evaluate the overall security improvement achieved and produce actionable recommendations.

1.5 Research Questions

- RQ1: What technical vulnerabilities, assessed against the OWASP Top Ten, are present in the baseline implementation of the Loyalty Card Management System?
- RQ2: To what extent can source-code-level remediation resolve the identified technical vulnerabilities, and how can this be verified?
- RQ3: What business-logic vulnerabilities exist in the loyalty-points and order-management workflows that a generic OWASP-based assessment would not detect?

- RQ4: How effectively can these business-logic vulnerabilities be mitigated through server-side enforcement and atomic database operations?

1.6 Scope of the Research

The assessment is conducted entirely within an isolated local environment (Kali Linux and XAMPP) using a purpose-built application, so that testing carries no risk to live systems or third-party data. The technical scope covers authentication, SQL injection, Cross-Site Scripting, Cross-Site Request Forgery, security headers, session-cookie configuration, input validation, and brute-force resistance. The business-logic scope covers price integrity during order placement, ownership enforcement on order cancellation, and atomicity of points redemption and order-completion transactions. Network-layer testing, operating-system hardening, and third-party dependency/supply-chain security are explicitly out of scope.

1.7 Dissertation Structure

Chapter 2 reviews existing literature on web application security, VAPT methodologies, and business-logic vulnerability research. Chapter 3 details the research methodology. Chapter 4 describes the design of the system under test. Chapter 5 presents the baseline security assessment. Chapter 6 evaluates the baseline results. Chapter 7 documents the remediation applied to every identified technical finding and the re-testing methodology used to verify it. Chapter 8 presents the business-logic vulnerability analysis that forms the project's principal novel contribution. Chapter 9 concludes the study and offers recommendations, and Chapter 10 provides a personal reflection on the project.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter reviews existing literature relevant to web application security, the OWASP Top Ten framework, VAPT methodologies, common security testing tools, security controls and remediation techniques, and - reflecting the extended scope of this project - business-logic vulnerabilities in web applications. The chapter closes by identifying the research gap that this project addresses.

2.2 Web Application Security and Common Threats

Web applications remain a primary target for attackers because they are internet-facing, handle sensitive data, and frequently contain implementation flaws introduced during rapid development [1]. Nawrocki and Kołodziej [5] observe that despite the maturity of secure-coding guidance, foundational vulnerabilities such as injection flaws, broken authentication and misconfiguration continue to appear in production systems, largely because security practices are inconsistently applied across development teams. Nair [6] similarly notes that phishing, XSS and SQL injection remain persistently common precisely because they exploit trust boundaries - between user input and application logic, and between application logic and the database - that are easy to specify correctly in principle but easy to omit in practice under delivery pressure.

Riskhan et al. [7] catalogue the most frequently observed real-world web application vulnerabilities and find that request-forgery and session-management weaknesses, while less discussed than injection flaws, are disproportionately damaging because they target already-authenticated users and privileged operations. This observation is directly relevant to the present study, in which the most severe baseline finding was a CSRF vulnerability affecting administrative account deletion - a privileged, authenticated operation.

2.3 OWASP Top Ten as a Security Framework

The OWASP Top Ten provides a widely adopted, consensus-based classification of the most critical web application security risks and is used throughout this project as the primary structuring framework for the baseline assessment. Odeh and Hijazi [8] argue that the framework's strength lies in its focus on impact and prevalence rather than purely technical novelty, which makes it well suited to guiding a systematic assessment such as the one conducted here. However, several authors note limitations relevant to this project. Ravindran and Potukuchi [4] observe that OWASP-aligned testing is oriented toward technical vulnerability classes and can under-represent application-specific business-logic flaws, which do not fit neatly into categories such as injection or broken access control despite frequently sharing an underlying cause - insufficient server-side trust verification. This limitation directly motivates Chapter 8 of this dissertation, which extends the assessment beyond the OWASP Top Ten into loyalty-platform-specific business logic.

Rohmaniah et al. [9] demonstrate the practical application of OWASP-Top-Ten-guided VAPT to a real website and confirm that CSRF and security-misconfiguration findings, though categorised as 'lower-profile' than injection vulnerabilities, are frequently the most exploitable in practice because they require no bypass of input sanitisation at all.

2.4 Vulnerability Assessment and Penetration Testing (VAPT) Methodologies

VAPT combines the breadth of automated vulnerability scanning with the depth of manual exploitation testing. Alquwayzani, Aldossri and Frikha [3] present VAPT as a structured, repeatable process for identifying and evaluating vulnerabilities in firewalls and web applications, and stress that automated results should always be manually verified before being treated as confirmed findings - a principle applied throughout the baseline assessment in Chapter 5 of this dissertation, where every automated ZAP alert was independently reproduced using Burp Suite before being classified as a genuine vulnerability.

A recurring theme in the literature is that VAPT is most effective when treated as a continuous activity integrated into the development lifecycle rather than a one-off exercise. Priambodo, Rifansyah and Hasbi [10] further argue that findings should be prioritised by business impact, not merely by count, a principle this project applies by treating the administrative-account CSRF vulnerability as the highest-priority remediation item despite the total number of findings being small.

Comparatively fewer studies address the remediation and re-verification stage of the VAPT lifecycle in similar depth. Joshi et al. [11] note in a survey of cloud-hosted VAPT research that most published studies report on the assessment phase only, with remediation typically left as a recommendation rather than an implemented and re-tested outcome. This project addresses that gap directly by implementing every recommended control at source-code level and documenting the resulting re-testing methodology in Chapter 7.

2.5 Security Testing Tools

OWASP ZAP is an open-source dynamic application security testing (DAST) tool widely used for both automated spidering/scanning and manual proxy-based interception. Automated active-scan rules are effective at surfacing candidate issues such as missing headers, but are comparatively weak at detecting logic-dependent vulnerabilities such as CSRF on specific business operations, reinforcing the necessity of manual verification with a tool such as Burp Suite.

SQLMap remains a standard tool for automated SQL injection testing, capable of testing a far larger permutation of injection techniques than would be practical manually. In this project, SQLMap was used to corroborate the manual finding that the authentication endpoints correctly used parameterised queries and were not exploitable via SQL injection.

Burp Suite Community Edition supports interception, modification and replay of HTTP requests, making it the primary tool for manually verifying request-forgery, business-logic and access-control findings in this project [12]. No single testing tool provides complete coverage; the combination of automated scanning for breadth and manual proxy-based testing for depth, as adopted in this project, reflects consensus best practice.

2.6 Security Controls and Remediation Techniques

Anti-CSRF tokens - unpredictable, session-bound values validated on every state-changing request - remain the standard defence against request-forgery attacks [13]. Gandikota et al. [14] emphasise that CSRF protection must be applied consistently across every state-changing endpoint, since a single

unprotected form (as was found in the administrative deletion function of this project's baseline system) is sufficient to compromise an otherwise well-protected application.

Session-cookie hardening through the HttpOnly, Secure and SameSite attributes is widely recommended to reduce the impact of both XSS and CSRF [15]. HttpOnly prevents client-side script access to the session cookie, Secure restricts transmission to encrypted connections, and SameSite restricts the circumstances under which the cookie is attached to cross-site requests, providing defence-in-depth alongside token-based CSRF protection rather than replacing it.

HTTP security headers - Content-Security-Policy, X-Frame-Options, X-Content-Type-Options and Referrer-Policy - constitute a further layer of browser-enforced protection against clickjacking, MIME-sniffing and content-injection attacks [16]. Missing security headers remain among the most commonly observed misconfigurations in student and small-organisation web applications, consistent with the baseline finding of this project.

Credential and secrets management is a further recurring theme. Dencheva [17] highlights that hard-coded database credentials, particularly the use of a privileged database account for routine application queries, materially increase the impact of any subsequent injection or misconfiguration vulnerability, since a compromise of the application immediately grants a compromise of the entire database server rather than a scoped subset of it. This principle directly informed the remediation applied in this project, where the application's hard-coded root-with-blank-password connection was replaced with a dedicated, least-privilege database account restricted to the operations the application actually requires.

2.7 Business-Logic Vulnerabilities in Web Applications

A smaller but growing body of literature addresses vulnerabilities that arise not from a failure to sanitise input, but from a failure to enforce the intended business rules of an application at the server. Kollepalli et al. [18] describe such flaws as arising when an application's security controls are technically functioning correctly (authentication succeeds, encryption is in place) while the workflow itself contains an exploitable assumption - for example, trusting a client-supplied value that should be authoritative only on the server. Because these vulnerabilities do not correspond to a generic payload pattern, they are largely invisible to automated scanners such as ZAP and require manual, domain-aware testing to uncover [19].

For platforms that manage a stored-value economy - loyalty points, wallet balances, or store credit - business-logic testing is particularly important because the 'currency' being protected is entirely represented in application state rather than an external payment processor, and its integrity depends entirely on the application's own transaction logic being both authorisation-checked and atomic [20]. Two classes of business-logic flaw are especially relevant to such platforms and form the basis of Chapter 8 of this dissertation. First, price-tampering vulnerabilities arise when an application derives a chargeable amount, or a resulting entitlement such as loyalty points, from a value supplied in the client request rather than re-deriving it from an authoritative server-side source; Bakır [21] notes that such flaws are trivially exploitable using an intercepting proxy such as Burp Suite once a request has been captured. Second, Insecure Direct Object Reference (IDOR) flaws arise when an application accepts an identifier (such as an order ID) from the client without verifying that the identifier belongs to the requesting user; Omotosho et al. [22] identify IDOR as one of the most common access-control failures

in transactional web applications precisely because ownership checks are easy to omit when an endpoint is otherwise correctly authenticated.

A further class of business-logic risk arises from race conditions in concurrent request handling. Zhang, Xing and Li [23] describe how an application that checks a condition (such as a sufficient points balance) and then separately acts on it (deducting points and issuing a reward) in two non-atomic steps can be exploited by submitting multiple near-simultaneous requests, allowing the condition to be satisfied more than once before the first deduction is committed - a double-spending pattern. The standard mitigation, reflected in the remediation applied in Chapter 8 of this dissertation, is to combine the check and the state change into a single atomic database operation whose WHERE clause re-asserts the precondition, relying on the database's own row-locking guarantees rather than application-level sequencing [12].

2.8 Conceptual Framework

This project adopts a two-layer conceptual framework. The first layer is the conventional OWASP-aligned VAPT lifecycle - reconnaissance, automated scanning, manual verification, risk classification, remediation and re-testing - applied to the technical vulnerability classes described in Sections 2.3 to 2.6. The second layer, addressing the gap identified in Section 2.7, is a business-logic assessment layer applied specifically to the loyalty-points and order-management workflows, following a check-identify-demonstrate-mitigate pattern for each candidate flaw. The two layers are integrated in this dissertation by applying them sequentially to the same system: Chapters 5 to 7 address the first layer, and Chapter 8 addresses the second.

2.9 Research Gap

The reviewed literature establishes that OWASP-based VAPT is a mature and widely validated methodology for identifying conventional technical vulnerabilities, and that anti-CSRF tokens, hardened session cookies, security headers and least-privilege credential management are well-established remediation techniques. Two gaps are nonetheless evident. First, as Joshi et al. [11] note, published VAPT case studies more commonly report assessment findings than implemented and re-verified remediation, leaving an incomplete picture of whether recommended fixes are straightforward to apply correctly in practice. Second, as argued by Ravindran and Potukuchi [4] and reinforced by Kollepalli et al. [18], generic OWASP-based assessment under-addresses business-logic vulnerabilities that are specific to an application's domain, such as the points-economy risks relevant to a loyalty-reward platform. This project addresses both gaps directly: by implementing and documenting complete source-level remediation for every baseline finding (Chapter 7), and by extending the assessment into a domain-specific business-logic layer that a generic OWASP assessment would not surface (Chapter 8).

2.10 Summary

This chapter has reviewed literature on web application security threats, the OWASP Top Ten framework, VAPT methodologies, common testing tools, established remediation techniques, and business-logic vulnerabilities. The review identifies a clear rationale for this project's two-layer

approach - conventional OWASP-aligned assessment and remediation, extended with a domain-specific business-logic analysis - which is developed across the remaining chapters.

CHAPTER 3: RESEARCH METHODOLOGY

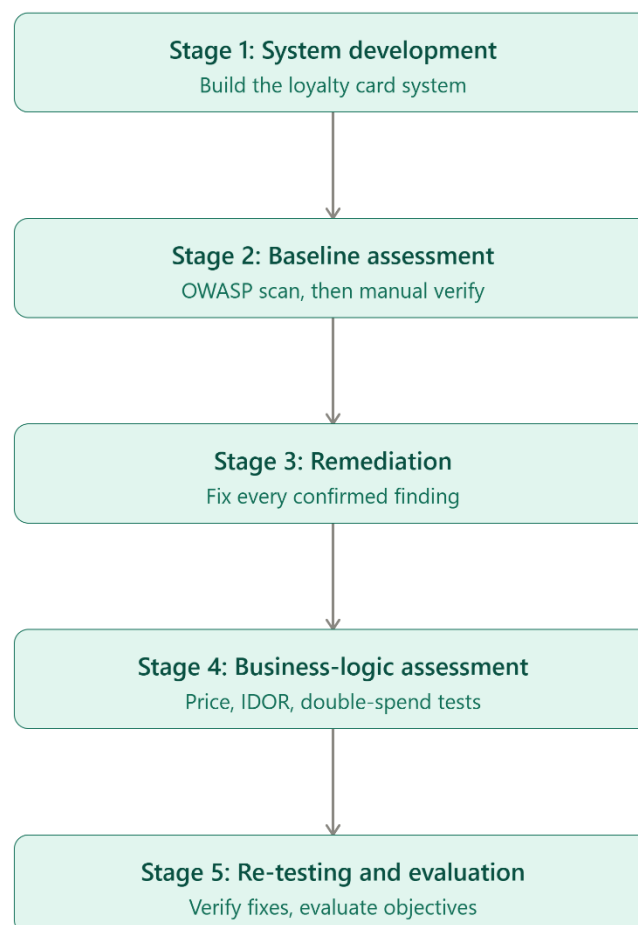
3.1 Introduction

This chapter describes the research methodology adopted for the project, including the overall research approach, research design, tools and techniques, experimental setup, and ethical considerations.

3.2 Research Approach

A pragmatic, quantitative experimental research approach is adopted. The study does not seek to test a general theory through statistical inference across a population of applications; rather, it applies an established methodology (VAPT guided by the OWASP Top Ten, extended with business-logic testing) to a single system under controlled conditions, and measures the outcome through concrete, reproducible evidence - captured requests, scanner alerts, and before/after comparisons of specific security controls. This approach is consistent with applied security research more broadly, where case-study-based experimental testing against a real or representative system is the established norm [16].

3.3 Research Design



3.4 Tools and Techniques

Tool	Purpose	Testing Stage
OWASP ZAP	Automated spidering, active scanning, and proxy-based interception	Baseline assessment, re-testing
Nikto	Automated web-server misconfiguration and known-vulnerability scanning	Baseline assessment
SQLMap	Automated SQL injection testing across multiple injection techniques	Baseline assessment
Burp Suite Community Edition	Manual request interception, modification and replay	Baseline assessment, business-logic testing, re-testing
Browser Developer Tools	Inspection of cookies, response headers and client-side behaviour	Baseline assessment, re-testing
phpMyAdmin	Database schema inspection and verification of server-side state changes	Business-logic testing

3.5 Experimental Setup

The system under test runs on a XAMPP stack (Apache, PHP, MariaDB/MySQL) hosted on a Windows machine, ensuring that the application never leaves a controlled local environment. Testing is conducted from a separate Kali Linux virtual machine running under Oracle VirtualBox in NAT networking mode, which reaches the host application via the VirtualBox NAT gateway address, replicating a realistic attacker-perspective network boundary between the testing platform and the target application while remaining fully isolated from any production network. This separation between target and testing platform mirrors real-world VAPT engagements, where the tester's toolchain is logically distinct from the system under assessment.

3.6 Ethical Considerations

All testing is conducted exclusively against a system developed specifically for this project, hosted entirely within an isolated local environment with no connection to production infrastructure or real user data. No third-party systems are targeted at any stage of the assessment. Test accounts, rather than real personal data, are used throughout. This approach ensures the project can be conducted without institutional ethics review beyond standard departmental approval, consistent with guidance that penetration testing research should be confined to systems for which explicit authorisation - in this case, ownership - exists.

3.7 Summary

This chapter has described the pragmatic, quantitative experimental methodology adopted for the project, its five-stage research design spanning system development, baseline assessment, remediation, business-logic assessment and re-testing, the tools used at each stage, the experimental environment, and the ethical considerations governing the work. The following chapter describes the design of the system under test.

CHAPTER 4: DESIGN PROCESS

4.1 Introduction

This chapter describes the design and implementation of the Loyalty Card Management System used as the subject of the security assessment, together with the experimental environment in which the assessment was conducted. The system was deliberately built to be functionally representative of a real loyalty/rewards platform - covering customer registration, authentication, a points dashboard, a shop and order workflow, rewards redemption, and an administrative panel - so that the resulting assessment and remediation would be applicable to real loyalty-based e-commerce systems rather than an artificially simplified test harness.

4.2 System Overview and Features

The system is implemented in PHP with a MySQL/MariaDB backend, following a traditional server-rendered architecture in which each page performs its own authentication check, database interaction and HTML rendering. This architecture was chosen because it is representative of a very large population of small-to-medium business web applications still in active production use, making the assessment's findings broadly generalisable.

Customer-facing functionality comprises: registration with email verification, login with brute-force lockout, a password-reset flow, a loyalty dashboard displaying the customer's card number, points balance and membership tier, a shop from which products can be ordered, an order-history view supporting cancellation of pending orders, a rewards catalogue supporting points redemption for single-use voucher codes, a points-history ledger, and profile and password management.

Administrative functionality comprises: an overview dashboard, customer search and points/status management, order completion (the point at which loyalty points are awarded), a voucher-code checker for redeeming issued vouchers, and management of the product and reward catalogues. Every state-changing administrative and customer action is recorded in an append-only audit log, and the underlying database connection uses a dedicated, least-privilege account rather than a shared administrative credential.

4.3 Experimental Environment

The application is hosted on a XAMPP stack (Apache 2.4, PHP 8, MariaDB) running on a Windows host. Figure 4.1 shows the XAMPP control panel confirming that the Apache and MySQL services required by the application are both running.

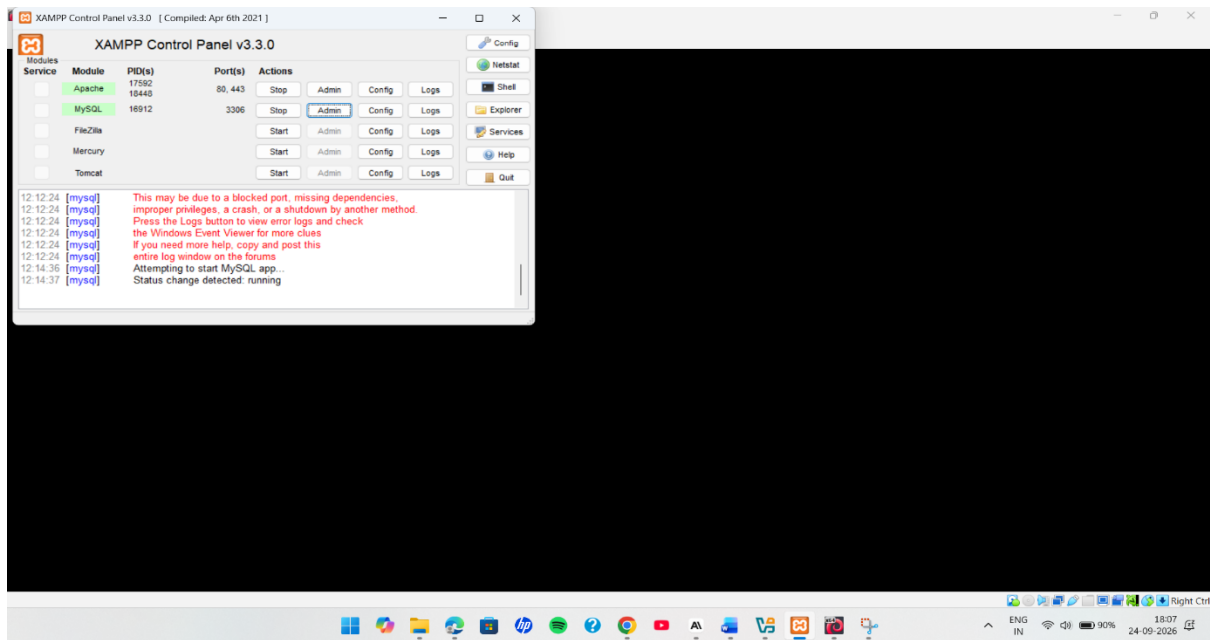


Figure 4.1: XAMPP Server Configuration and Service Status

Figure 4.2 shows the resulting database structure in phpMyAdmin, comprising the tables that support the full feature set described in Section 4.2, including users, orders, products, rewards, redemptions, points_ledger, audit_log, and the rate-limiting and password-reset support tables.

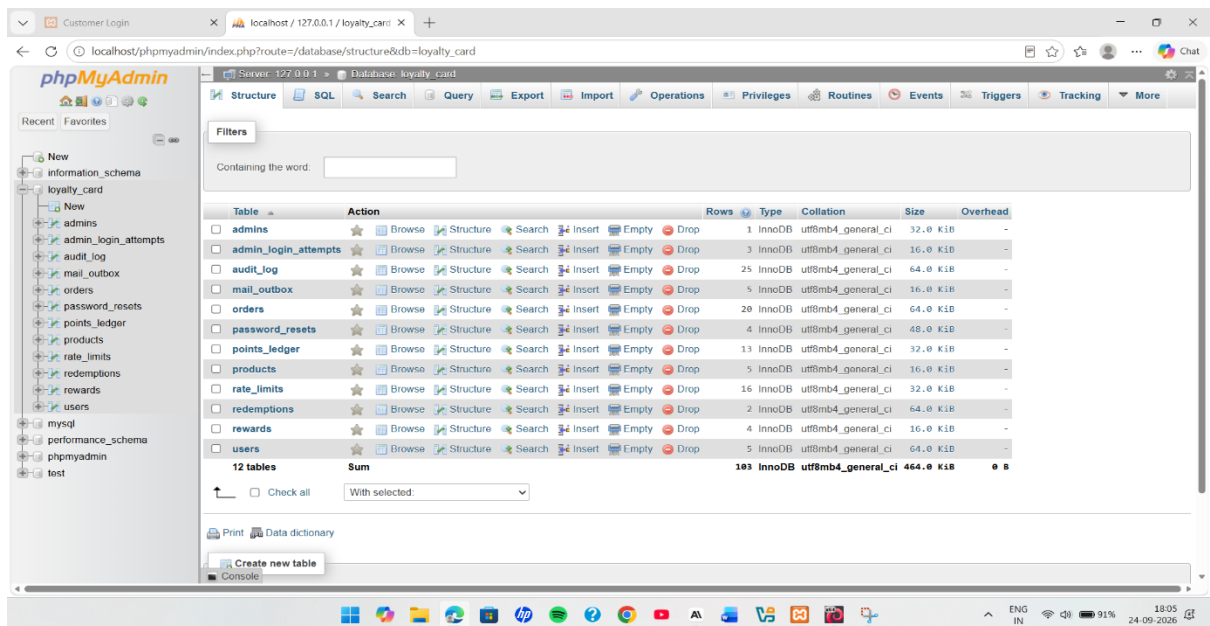


Figure 4.2: Loyalty Card Database Structure in phpMyAdmin

4.4 Application Interfaces

The following figures illustrate the principal customer- and administrator-facing interfaces of the system as implemented prior to the security assessment described in Chapter 5.

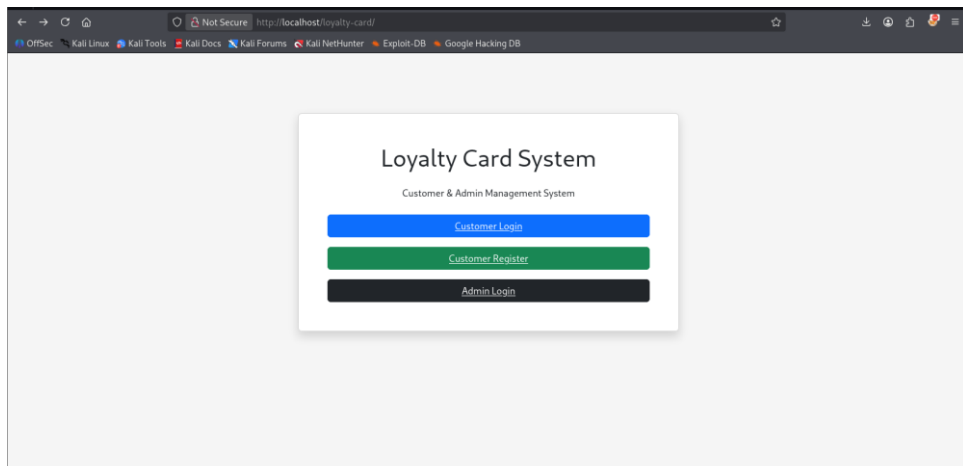


Figure 4.3: Loyalty Card System Main Interface

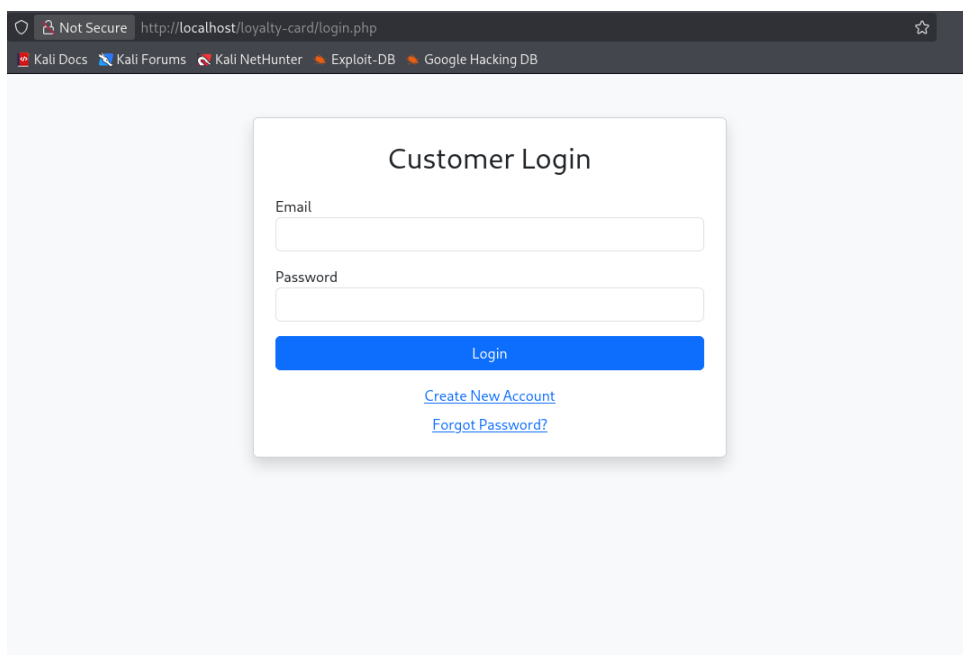
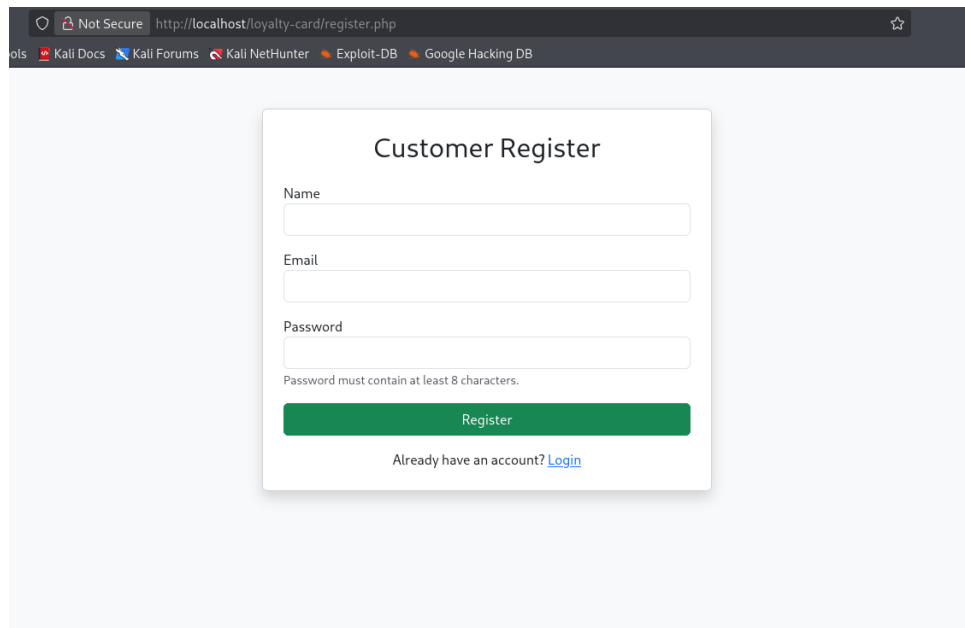


Figure 4.4: Customer Login Interface



A screenshot of a web browser displaying the 'Customer Register' form. The browser's address bar shows 'http://localhost/loyalty-card/register.php'. The form is centered on a light gray background and contains the following elements: a title 'Customer Register', three input fields for 'Name', 'Email', and 'Password', a note 'Password must contain at least 8 characters.', a green 'Register' button, and a link 'Already have an account? [Login](#)'.

Customer Register

Name

Email

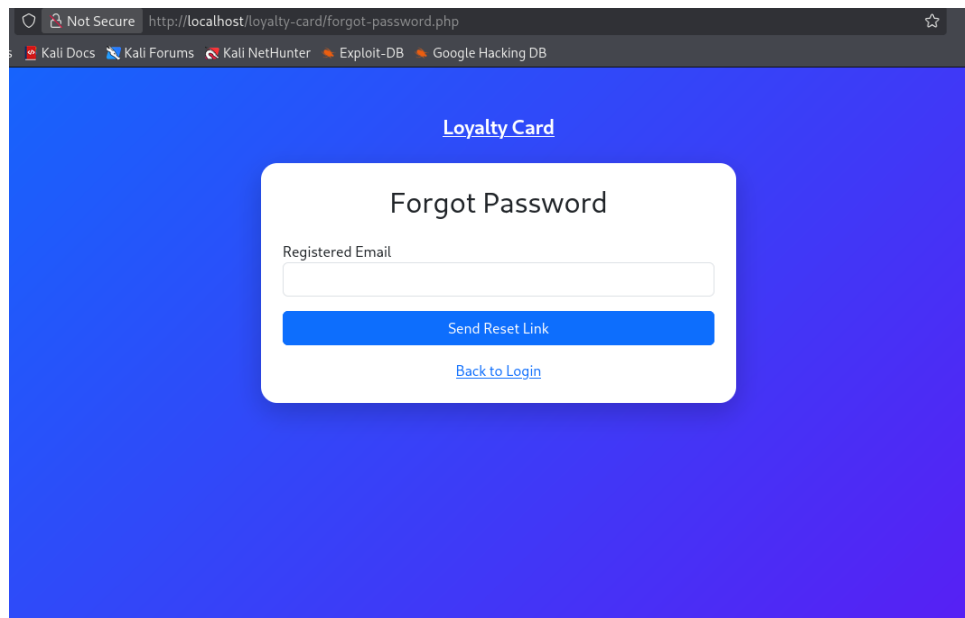
Password

Password must contain at least 8 characters.

Register

Already have an account? [Login](#)

Figure 4.5: Customer Registration Interface



A screenshot of a web browser displaying the 'Forgot Password' form. The browser's address bar shows 'http://localhost/loyalty-card/forgot-password.php'. The form is centered on a blue gradient background and contains the following elements: a title 'Forgot Password', an input field for 'Registered Email', a blue 'Send Reset Link' button, and a link '[Back to Login](#)'.

Loyalty Card

Forgot Password

Registered Email

Send Reset Link

[Back to Login](#)

Figure 4.6: Forgot Password Interface

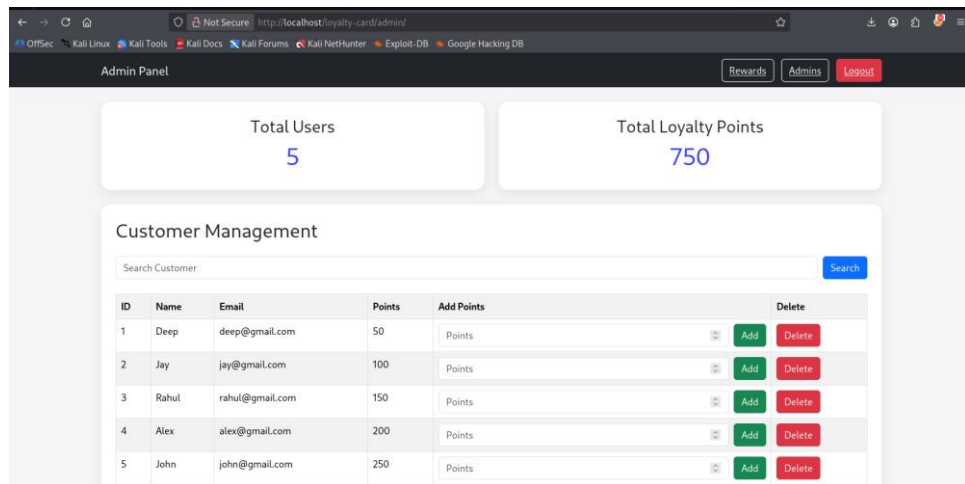


Figure 4.7: Administrator Dashboard and Customer Management Interface

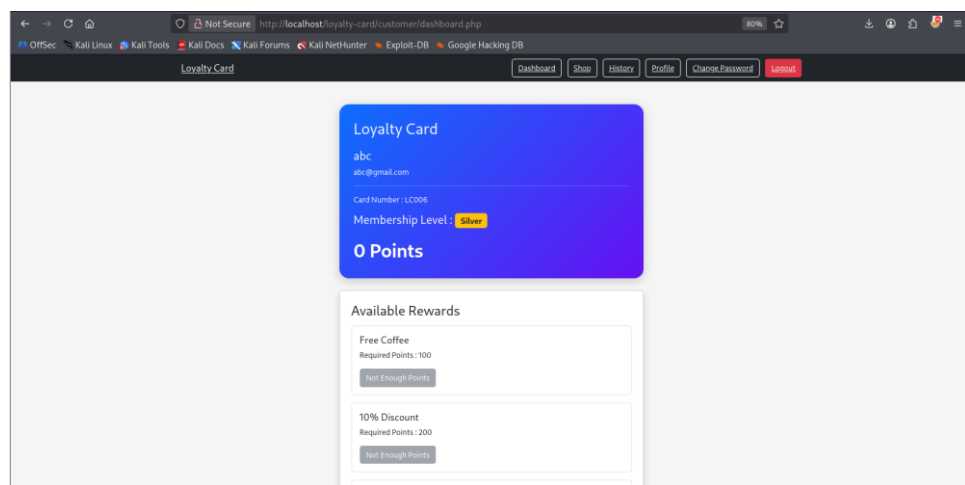


Figure 4.8: Customer Dashboard and Loyalty Rewards Interface

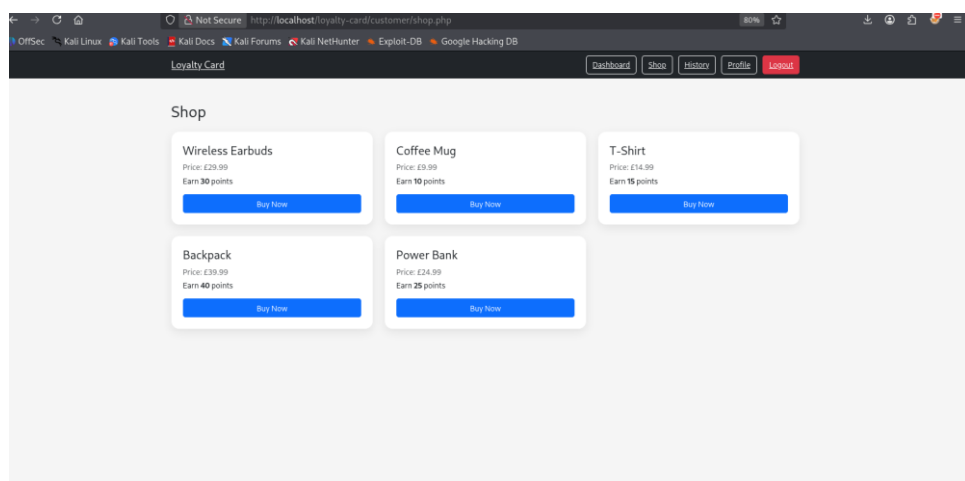


Figure 4.9: Customer Shop and Product Purchase Interface

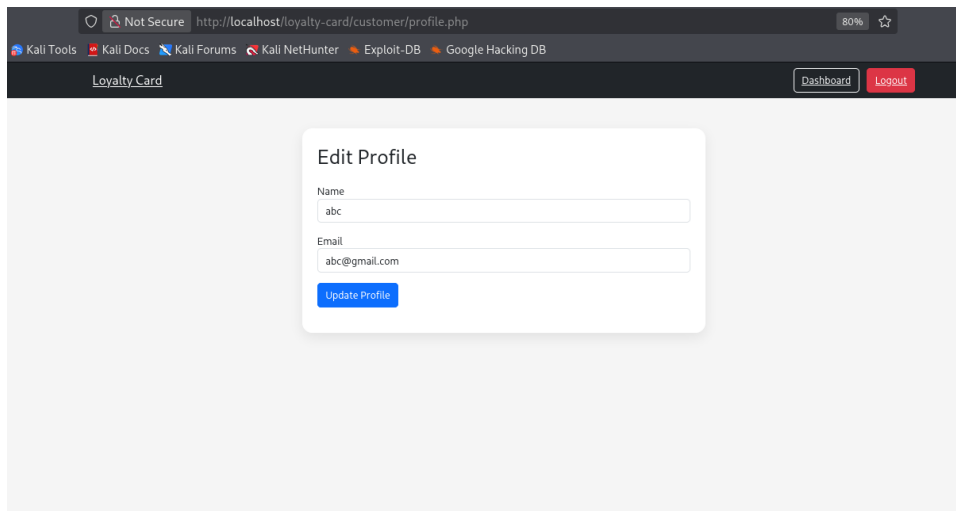


Figure 4.10: Customer Profile Management Interface

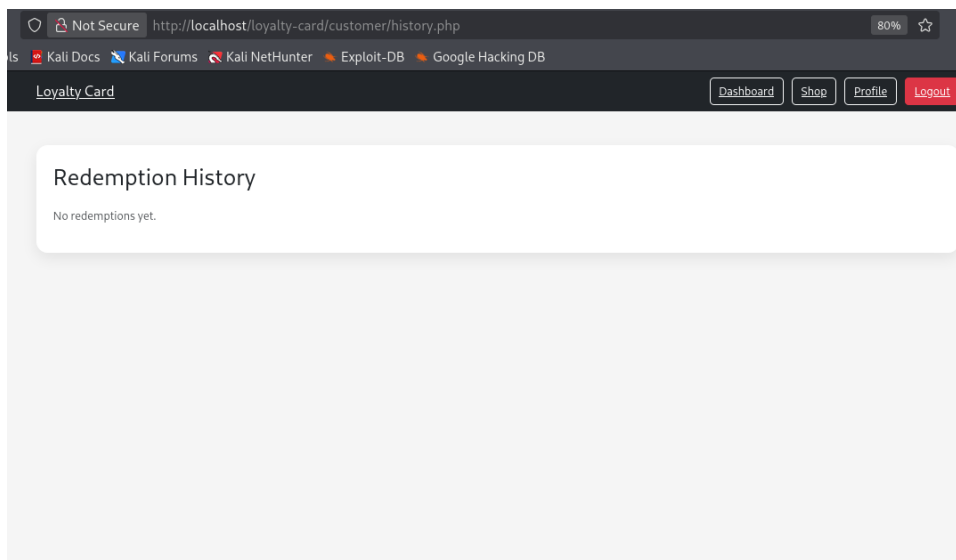


Figure 4.11: Customer Redemption History Interface

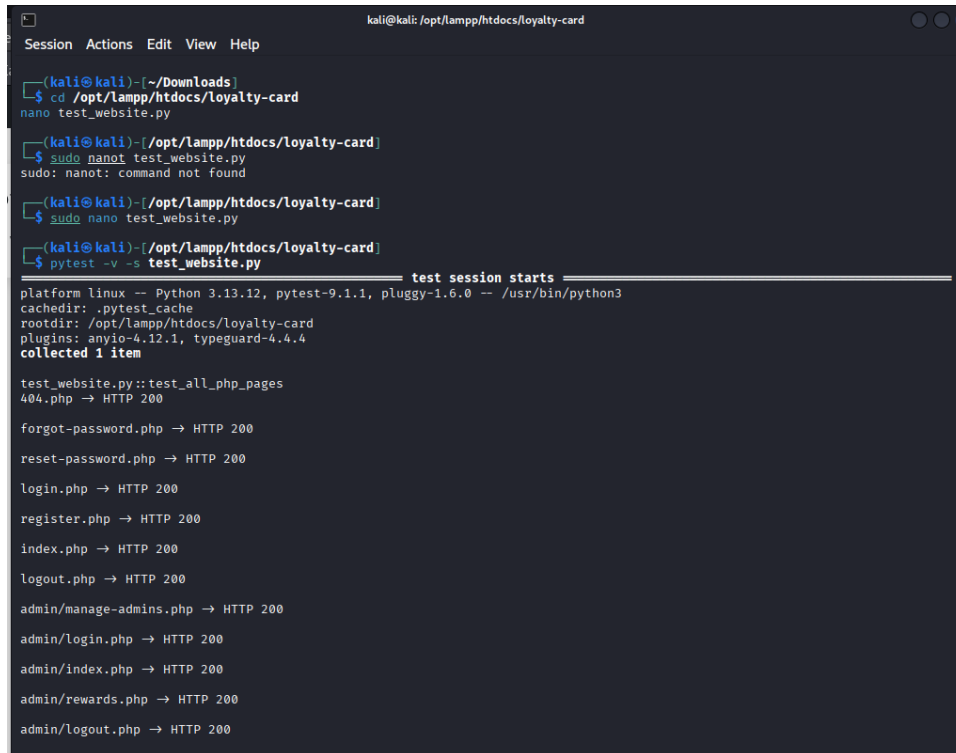
4.4A Design Rationale

Several design decisions were made deliberately to maximise the value of the subsequent security assessment rather than purely to minimise development effort. First, the administrative panel was given genuine privileged capability - directly adjusting a customer's points balance, locking accounts, and deleting other administrators - rather than being limited to read-only reporting, because privileged functionality of this kind is precisely where the consequences of a missing access-control or CSRF check are most severe, and therefore most instructive to assess. Second, the points-redemption and order-completion workflows were implemented as genuine, persisted state changes affecting a real database-backed balance, rather than being simulated, so that business-logic testing in Chapter 8 could be performed against an authentic transactional workflow rather than a stand-in. Third, the database schema was designed with support for the eventual security controls in mind from the outset - for example, a status column on customer accounts to support account locking, and a session-version column to support future session-invalidation functionality - so that the remediation programme in

Chapter 7 could be implemented without requiring disruptive schema changes partway through the project.

4.5 Pre-Assessment Functional Verification

Before beginning the security assessment, the availability of every application endpoint was verified using an automated page-availability check, followed by a Pytest-based endpoint validation pass, to confirm that the system was functionally stable and that any subsequent finding during security testing could be attributed to a security weakness rather than an unrelated functional defect.



```
kali@kali: /opt/lampp/htdocs/loyalty-card
Session Actions Edit View Help

(kali@kali)~/Downloads
$ cd /opt/lampp/htdocs/loyalty-card
nano test_website.py

(kali@kali)~/Downloads
$ sudo nanot test_website.py
sudo: nanot: command not found

(kali@kali)~/Downloads
$ sudo nano test_website.py

(kali@kali)~/Downloads
$ pytest -v -s test_website.py

===== test session starts =====
platform linux -- Python 3.13.12, pytest-9.1.1, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /opt/lampp/htdocs/loyalty-card
plugins: anyio-4.12.1, typeguard-4.4.4
collected 1 item

test_website.py::test_all_php_pages
404.php → HTTP 200

forgot-password.php → HTTP 200

reset-password.php → HTTP 200

login.php → HTTP 200

register.php → HTTP 200

index.php → HTTP 200

logout.php → HTTP 200

admin/manage-admins.php → HTTP 200

admin/login.php → HTTP 200

admin/index.php → HTTP 200

admin/rewards.php → HTTP 200

admin/logout.php → HTTP 200
```

Figure 4.12: Automated Website Page Availability Testing



```
admin/logout.php → HTTP 200

customer/dashboard.php → HTTP 200

customer/profile.php → HTTP 200

customer/shop.php → HTTP 200

customer/change-password.php → HTTP 200

customer/history.php → HTTP 200

includes/csrf.php → HTTP 200

includes/db.php → HTTP 200
PASSED

===== warnings summary =====
../usr/lib/python3/dist-packages/_pytest/cache/provider.py:469
/usr/lib/python3/dist-packages/_pytest/cache/provider.py:469: PytestCacheWarning: could not create cache path /opt/lampp/ht
docs/loyalty-card/.pytest_cache/v/cache/nodeids: [Errno 13] Permission denied: '/opt/lampp/htdocs/loyalty-card/.pytest-cache-
files-7nrethir'
  config.cache.set("cache/nodeids", sorted(self.cached_nodeids))

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 1 passed, 1 warning in 0.82s =====

(kali@kali)~/Downloads
$
```

Figure 4.13: Pytest Application Endpoint Validation Results

4.6 Summary

This chapter has described the design, feature set, and experimental environment of the Loyalty Card Management System, and confirmed its functional readiness for security assessment. The following chapter presents the baseline security assessment conducted against this implementation.

CHAPTER 5: BASELINE SECURITY ASSESSMENT

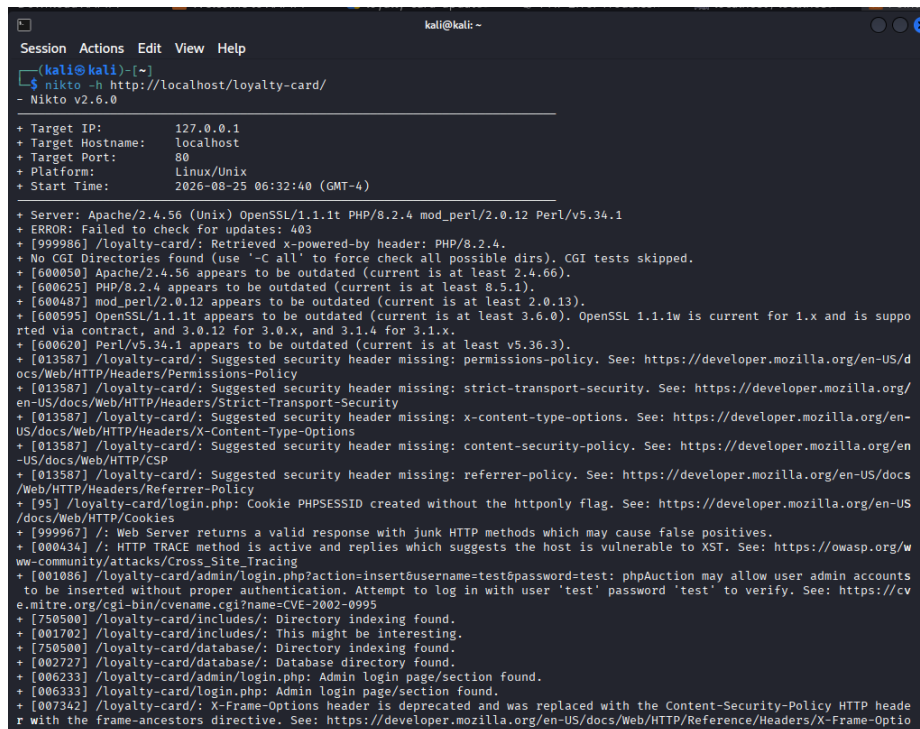
5.1 Introduction

This chapter presents the baseline security assessment conducted against the Loyalty Card Management System described in Chapter 4, prior to any remediation. The assessment follows the methodology set out in Chapter 3: automated scanning using OWASP ZAP and SQLMap, supported by manual verification using Burp Suite Community Edition, with every automated finding independently reproduced before being classified as a confirmed vulnerability. Testing progresses through reconnaissance, authentication testing, injection testing, client-side testing, request-forgery testing, and configuration assessment. A successful attack is reported as a confirmed vulnerability; a failed attack is reported as evidence of an existing, working security control.

Throughout this chapter, each test follows the same evidentiary standard applied consistently across the assessment: an automated alert or a suspected weakness is treated as unconfirmed until it has been independently reproduced through a manual, observable action - a request captured and replayed, a header inspected directly, or a payload traced through to its rendered output - so that every finding reported in Section 5.10 and evaluated in Chapter 6 rests on direct evidence rather than on tool output alone.

5.2 Scanning and Reconnaissance

The first phase establishes the application's attack surface. The locally hosted application was scanned to identify accessible endpoints, exposed resources, and server-level configuration issues.



```
kali@kali ~  
Session Actions Edit View Help  
--(kali@kali)~  
$ nikto -h http://localhost/loyalty-card/  
- Nikto v2.6.0  
  
+ Target IP: 127.0.0.1  
+ Target Hostname: localhost  
+ Target Port: 80  
+ Platform: Linux/Unix  
+ Start Time: 2026-08-25 06:32:40 (GMT-4)  
  
+ Server: Apache/2.4.56 (Unix) OpenSSL/1.1.1t PHP/8.2.4 mod_perl/2.0.12 Perl/v5.34.1  
+ ERROR: Failed to check for updates: 403  
+ [999986] /loyalty-card/: Retrieved x-powered-by header: PHP/8.2.4.  
+ No CGI Directories found (use '-C all' to force check all possible dirs). CGI tests skipped.  
+ [600050] Apache/2.4.56 appears to be outdated (current is at least 2.4.66).  
+ [600625] PHP/8.2.4 appears to be outdated (current is at least 8.5.1).  
+ [600487] mod_perl/2.0.12 appears to be outdated (current is at least 2.0.13).  
+ [600595] OpenSSL/1.1.1t appears to be outdated (current is at least 3.6.0). OpenSSL 1.1.1w is current for 1.x and is supported via contract, and 3.0.12 for 3.0.x, and 3.1.4 for 3.1.x.  
+ [600620] Perl/v5.34.1 appears to be outdated (current is at least v5.36.3).  
+ [013587] /loyalty-card/: Suggested security header missing: permissions-policy. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Permissions-Policy  
+ [013587] /loyalty-card/: Suggested security header missing: strict-transport-security. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security  
+ [013587] /loyalty-card/: Suggested security header missing: x-content-type-options. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options  
+ [013587] /loyalty-card/: Suggested security header missing: content-security-policy. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP  
+ [013587] /loyalty-card/: Suggested security header missing: referrer-policy. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy  
+ [95] /loyalty-card/login.php: Cookie PHPSESSID created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies  
+ [999967] //: Web Server returns a valid response with junk HTTP methods which may cause false positives.  
+ [000434] //: HTTP TRACE method is active and replies which suggests the host is vulnerable to XST. See: https://owasp.org/www-community/attacks/Cross_Site_Tracing  
+ [001086] /loyalty-card/admin/login.php?action=insert&username=test&password=test: phpAuction may allow user admin accounts to be inserted without proper authentication. Attempt to log in with user 'test' password 'test' to verify. See: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0999  
+ [750500] /loyalty-card/includes/: Directory indexing found.  
+ [001792] /loyalty-card/includes/: This might be interesting.  
+ [750500] /loyalty-card/database/: Directory indexing found.  
+ [002727] /loyalty-card/database/: Database directory found.  
+ [006233] /loyalty-card/admin/login.php: Admin login page/section found.  
+ [006333] /loyalty-card/login.php: Admin login page/section found.  
+ [007342] /loyalty-card/: X-Frame-Options header is deprecated and was replaced with the Content-Security-Policy HTTP header with the frame-ancestors directive. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/X-Frame-Options
```

Figure 5.1: Basic Website Scanning

The baseline web-server scan surfaced information made public by the running application and provided an overview of the available web service, used as the starting point for application-level testing.

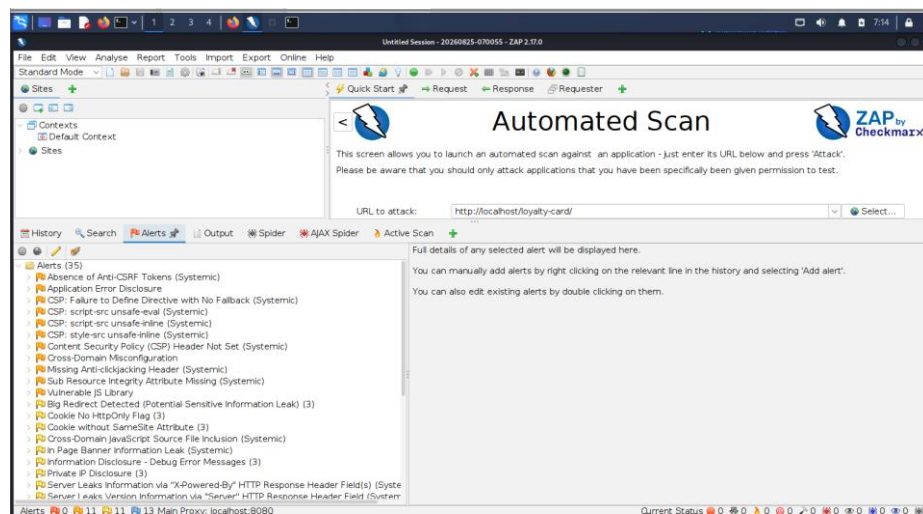


Figure 5.2: Discovered Vulnerabilities

The automated security assessment reported several potential issues, including a possible absence of anti-CSRF protection. Automated alerts of this kind are treated as signals to be manually verified rather than confirmed findings in their own right; the subsequent manual tests in this chapter determine whether the underlying state-changing operations can in fact be performed without adequate protection.

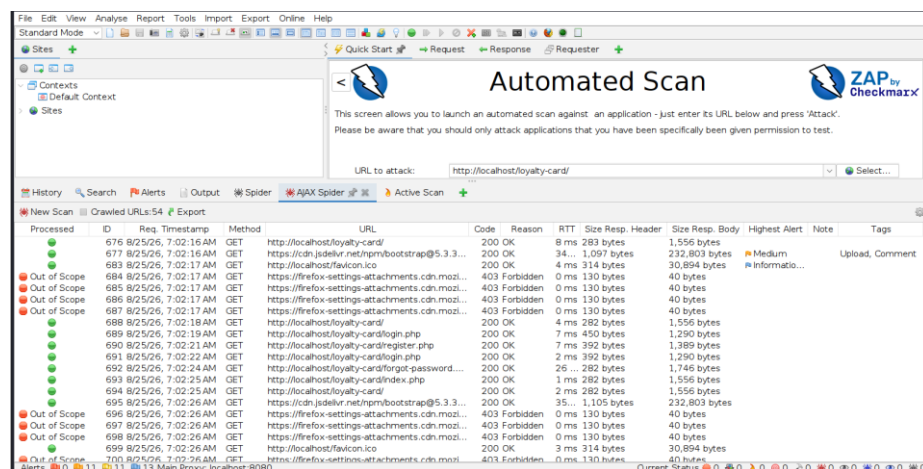


Figure 5.3: Crawled Endpoints

Crawling the application identified the accessible pages and endpoints that make up its functional structure, ensuring that both customer and administrative functionality were included in the subsequent evaluation.

The screenshot shows the ZAP by Checkmarx Automated Scan interface. The top section has a title 'Automated Scan' and a brief instruction: 'This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'. Please be aware that you should only attack applications that you have been specifically given permission to test.' Below this is a text input field for 'URL to attack:' containing 'http://localhost/loyalty-card/'. The bottom section displays a table of scan results.

ID	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
3.512	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/login.php	200	OK	5 ms	337 bytes	1,290 bytes
3.513	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/admin/login.php	200	OK	1 ms	337 bytes	2,063 bytes
3.514	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/register.php	200	OK	2 ms	337 bytes	1,389 bytes
3.515	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	GET	http://localhost/loyalty-card/login.php	200	OK	3 ms	365 bytes	1,290 bytes
3.516	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/admin/login.php	200	OK	10 ms	337 bytes	2,063 bytes
3.517	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/admin/login.php	200	OK	11 ms	337 bytes	2,063 bytes
3.518	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/login.php	200	OK	12 ms	337 bytes	1,290 bytes
3.519	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/register.php	200	OK	7 ms	337 bytes	1,389 bytes
3.520	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/admin/login.php	200	OK	6 ms	337 bytes	2,063 bytes
3.521	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/admin/login.php	200	OK	2 ms	337 bytes	2,063 bytes
3.522	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	GET	http://localhost/loyalty-card/login.php	200	OK	5 ms	365 bytes	1,290 bytes
3.523	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/admin/login.php	200	OK	5 ms	337 bytes	2,063 bytes
3.524	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/login.php	200	OK	2 ms	337 bytes	1,290 bytes
3.525	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/admin/login.php	200	OK	2 ms	337 bytes	2,063 bytes
3.526	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/admin/login.php	200	OK	5 ms	337 bytes	2,063 bytes
3.527	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	GET	http://localhost/loyalty-card/login.php	200	OK	4 ms	365 bytes	1,290 bytes
3.528	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/register.php	200	OK	4 ms	337 bytes	1,389 bytes
3.529	8/25/26, 7:04:23 AM	8/25/26, 7:04:23 AM	POST	http://localhost/loyalty-card/admin/login.php	200	OK	1 ms	337 bytes	2,063 bytes

Figure 5.4: Active Scanning Results

Active scanning analysed the application's responses to a range of security test requests, providing further candidate targets for manual validation. Automated results are treated throughout this assessment as a starting point rather than a conclusion, consistent with the methodology set out in Chapter 3.

5.3 Authentication Security Testing

Authentication testing establishes whether legitimate users can log in successfully and whether invalid credentials are correctly rejected, providing a reliable baseline before further testing.

The screenshot shows a web browser window with the address bar displaying 'http://localhost/loyalty-card/login.php'. The page content includes a 'Customer Login' form. The form has two input fields: 'Email' with the value 'abc@gmail.com' and 'Password' with masked characters '.....'. Below the password field is a blue 'Login' button. At the bottom of the form, there are two links: 'Create New Account' and 'Forgot Password?'. The browser's status bar at the bottom shows 'Kali Docs', 'Kali Forums', 'Kali NetHunter', 'Exploit-DB', and 'Google Hacking DB'.

Figure 5.5: Login Authentication Check

A normal customer login request using valid credentials is processed successfully, confirming that the authentication function operates correctly and returns an authenticated session for further testing.

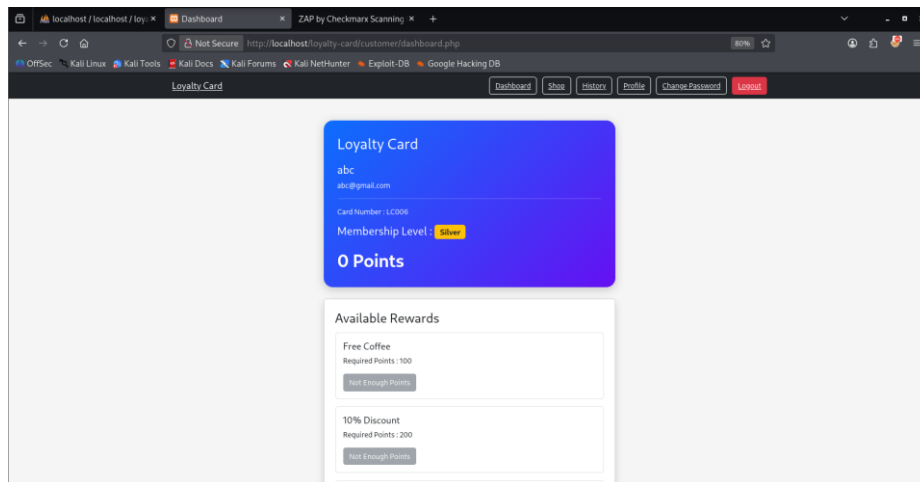


Figure 5.6: Authenticated Login Successful

Successful authentication correctly redirects the user to their dashboard, confirming that valid credentials are honoured and that access control operates as expected under normal conditions.

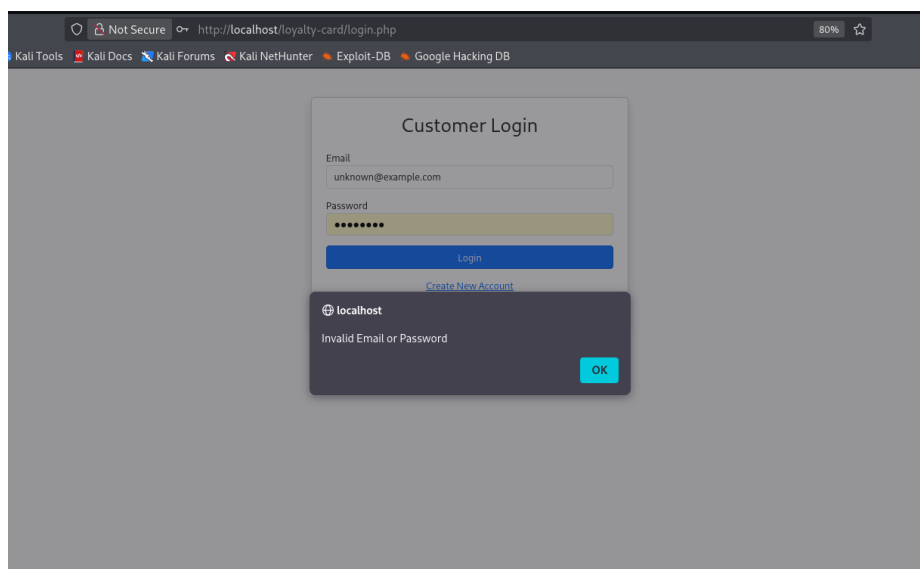


Figure 5.7: Incorrect Password Login Attempt

An incorrect password submitted to the same endpoint is correctly rejected, confirming that the login mechanism does not simply accept arbitrary input.

5.4 SQL Injection Testing

SQL injection testing determines whether user-controlled parameters can alter the application's SQL queries. The login function was selected as the primary target because it directly processes user-supplied authentication data; testing combined manual and automated techniques.

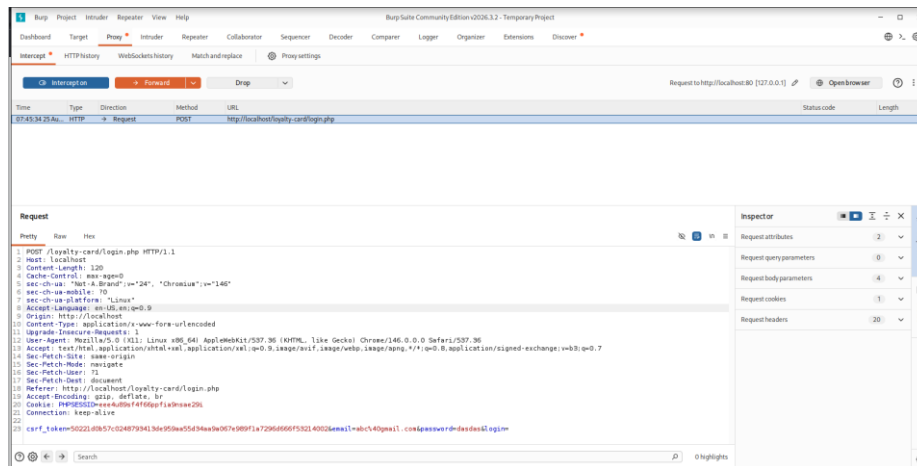


Figure 5.8: Captured Login Request

The login POST request was captured in Burp Suite to establish a controlled baseline request whose parameters could subsequently be modified for injection testing.

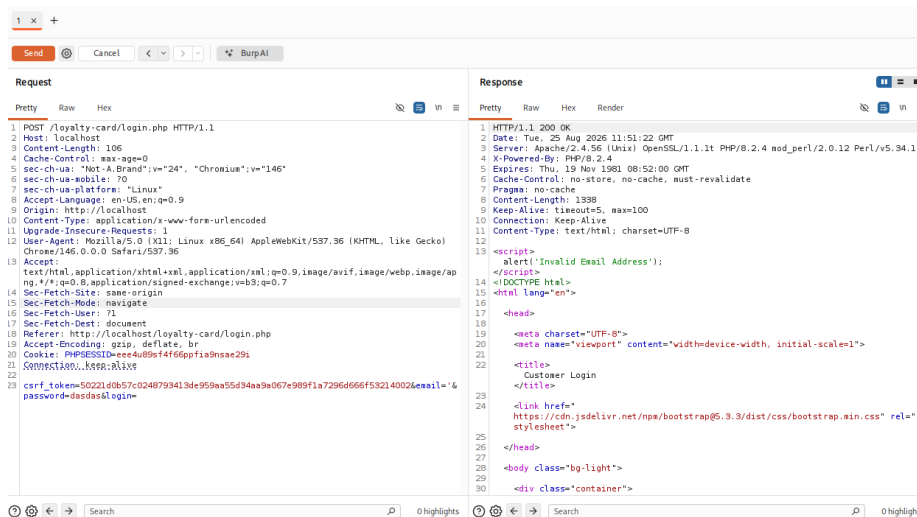


Figure 5.9: SQL Injection Payload Testing

A controlled SQL injection payload was substituted into the captured authentication request to determine whether it altered the application's database logic or allowed authentication to be bypassed.

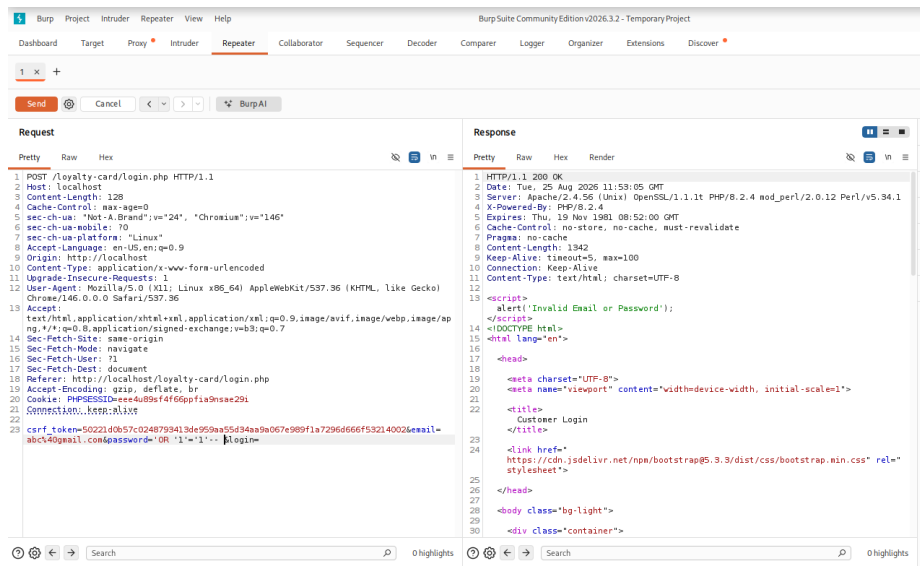


Figure 5.10: SQL Injection Attempt Failed

The manipulated request did not result in authentication bypass or any indication of database-level manipulation, consistent with the use of parameterised queries in the application's authentication logic.

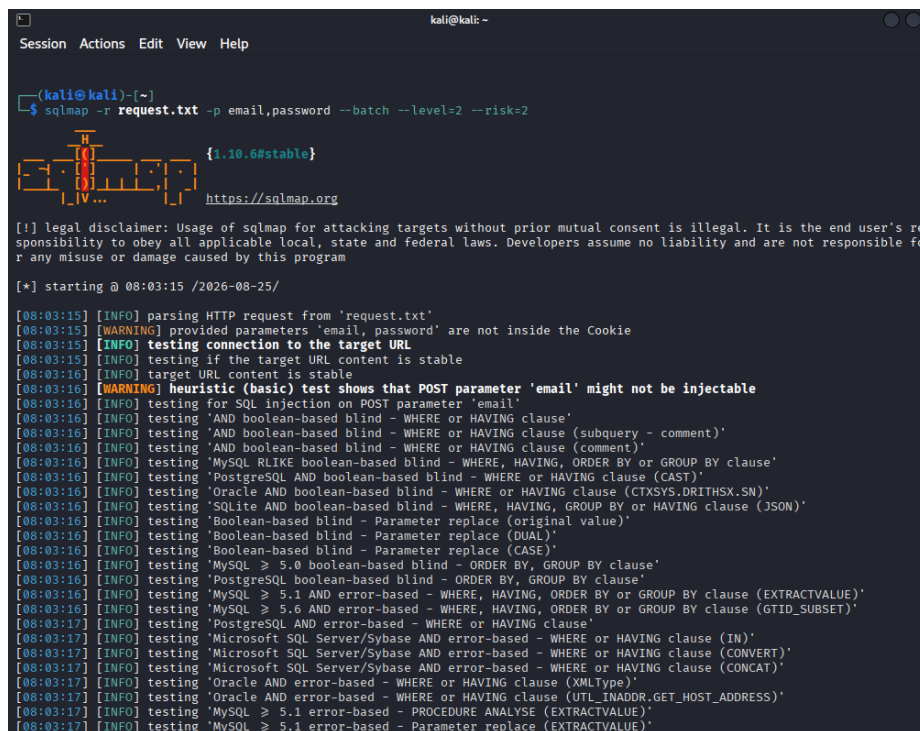


Figure 5.11: Automated Testing Using SQLMap

Automated SQL injection testing was subsequently performed against the same captured request using SQLMap, which tests a substantially larger set of injection techniques than is practical to reproduce manually.

```

kali@kali: ~
Session Actions Edit View Help
[08:03:25] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[08:03:26] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[08:03:27] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (CONVERT)'
[08:03:28] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (CONCAT)'
[08:03:29] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[08:03:30] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (UTL_INADDR.GET_HOST_ADDRESS)'
[08:03:31] [INFO] testing 'MySQL >= 5.1 error-based - PROCEDURE ANALYSE (EXTRACTVALUE)'
[08:03:31] [INFO] testing 'MySQL >= 5.1 error-based - Parameter replace (EXTRACTVALUE)'
[08:03:31] [INFO] testing 'PostgreSQL error-based - Parameter replace'
[08:03:32] [INFO] testing 'Microsoft SQL Server/Sybase error-based - Stacking (EXEC)'
[08:03:32] [INFO] testing 'Generic inline queries'
[08:03:32] [INFO] testing 'MySQL inline queries'
[08:03:32] [INFO] testing 'PostgreSQL inline queries'
[08:03:32] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[08:03:32] [INFO] testing 'Oracle inline queries'
[08:03:33] [INFO] testing 'MySQL >= 5.0.12 stacked queries (comment)'
[08:03:33] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[08:03:34] [INFO] testing 'PostgreSQL stacked queries (heavy query - comment)'
[08:03:34] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[08:03:35] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (DECLARE - comment)'
[08:03:36] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[08:03:36] [INFO] testing 'Oracle stacked queries (heavy query - comment)'
[08:03:37] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[08:03:38] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (SLEEP)'
[08:03:38] [INFO] testing 'MySQL < 5.0.12 AND time-based blind (BENCHMARK)'
[08:03:39] [INFO] testing 'MySQL >= 5.0.12 RLIKE time-based blind'
[08:03:40] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[08:03:41] [INFO] testing 'PostgreSQL AND time-based blind (heavy query)'
[08:03:42] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[08:03:42] [INFO] testing 'Microsoft SQL Server/Sybase AND time-based blind (heavy query)'
[08:03:43] [INFO] testing 'Oracle AND time-based blind'
[08:03:44] [INFO] testing 'Oracle AND time-based blind (heavy query)'
[08:03:45] [INFO] testing 'Informix AND time-based blind (heavy query)'
[08:03:46] [INFO] testing 'MySQL >= 5.0.12 time-based blind - Parameter replace'
[08:03:46] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[08:03:47] [INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'
[08:03:48] [WARNING] POST parameter 'password' does not seem to be injectable
[08:03:49] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' op
tions if you wish to perform more tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF)
maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch '--random-agent'

[*] ending @ 08:03:49 /2026-08-25/

kali@kali: ~
$

```

Figure 5.12: SQLMap - No Injectable Parameter Found

SQLMap did not identify an injectable parameter in either the email or password fields of the tested login request. Combining the manual and automated results, SQL injection was not demonstrated under the tested conditions, and this finding is treated as confirmation of an effective, existing security control (parameterised queries) rather than the absence of a vulnerability worth testing.

5.5 Cross-Site Scripting (XSS) Testing

Cross-Site Scripting testing determines whether user-controlled content is rendered by the browser as executable script rather than as inert text. A representative payload was submitted through the customer profile-update function, which subsequently displays the stored value back to the user.

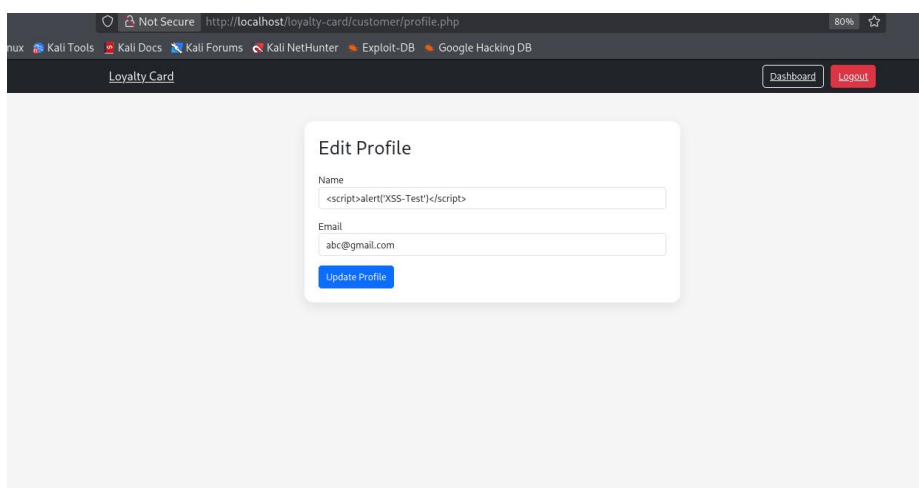


Figure 5.13: Inputting an XSS Payload on Profile

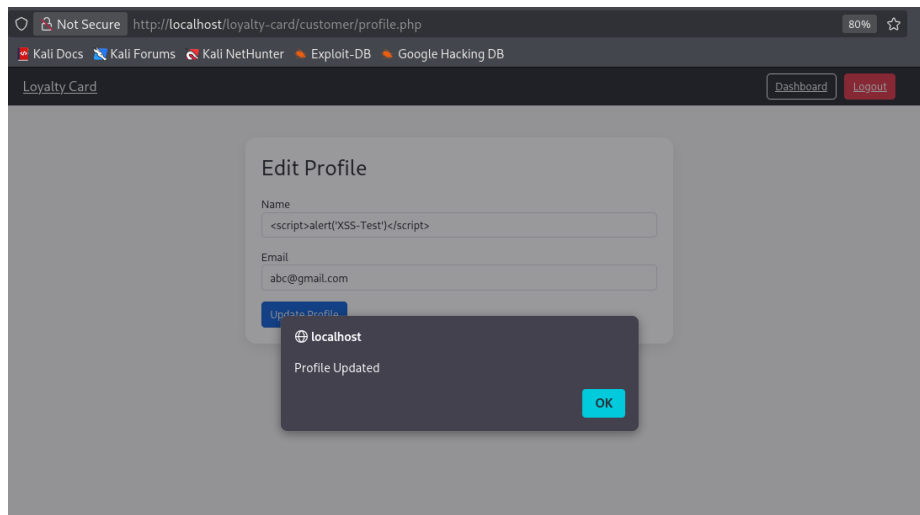


Figure 5.14: XSS Payload Did Not Execute

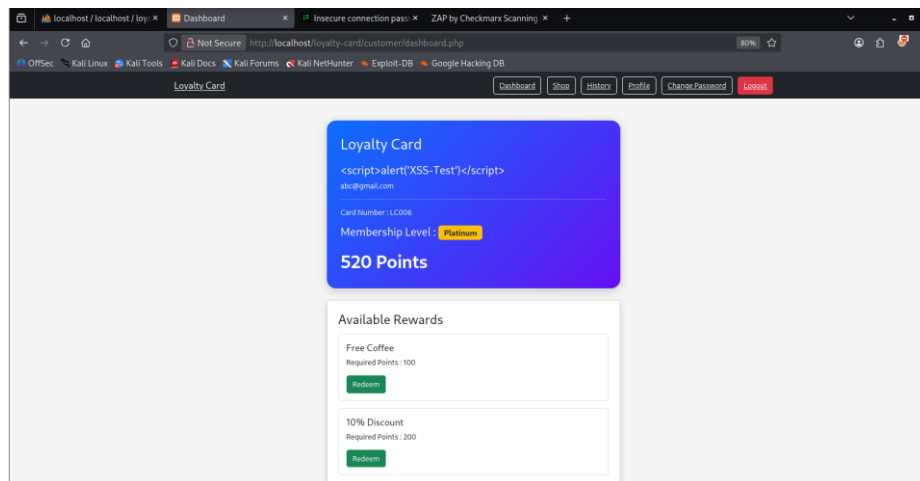


Figure 5.15: Value Rendered Without XSS

In each case the submitted payload was rendered as plain text rather than being interpreted as HTML or JavaScript, indicating that output encoding was correctly applied to this field. This is a meaningful result because the presence of a malicious-looking string in stored input does not, by itself, constitute a confirmed vulnerability - it must actually execute to be classified as exploitable, and it did not.

5.6 Cross-Site Request Forgery (CSRF) Testing

CSRF testing determines whether authenticated, state-changing operations can be triggered without a valid, server-verified request token. Three functions were tested because a successful exploit against any of them could alter sensitive account data or a privileged account: profile modification, password modification, and administrative account deletion.

5.6.1 Profile Modification

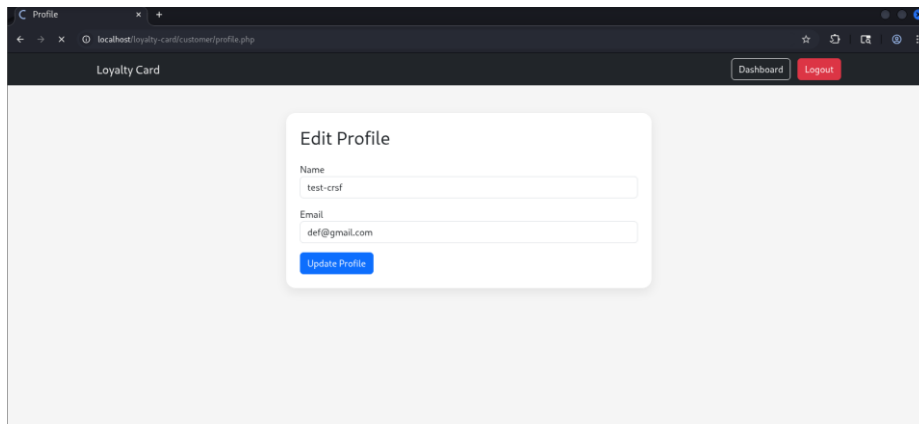


Figure 5.16: Updating the Profile

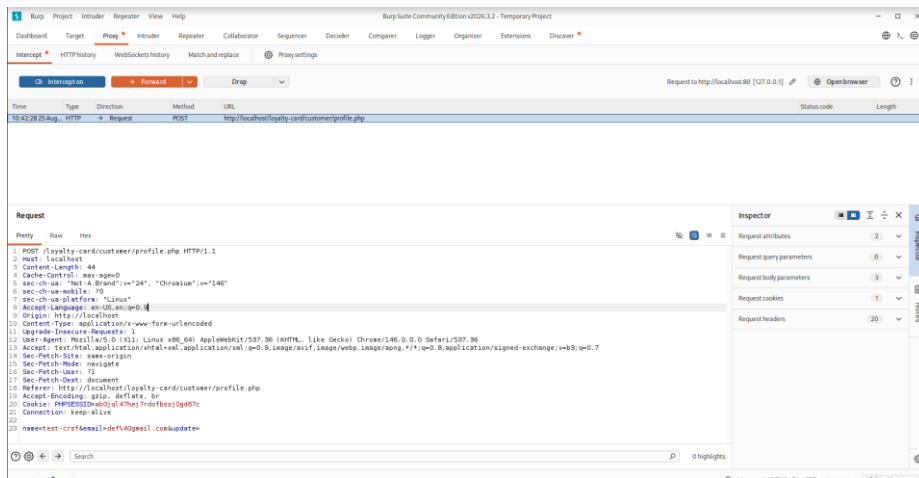


Figure 5.17: Capturing the Profile Update Request

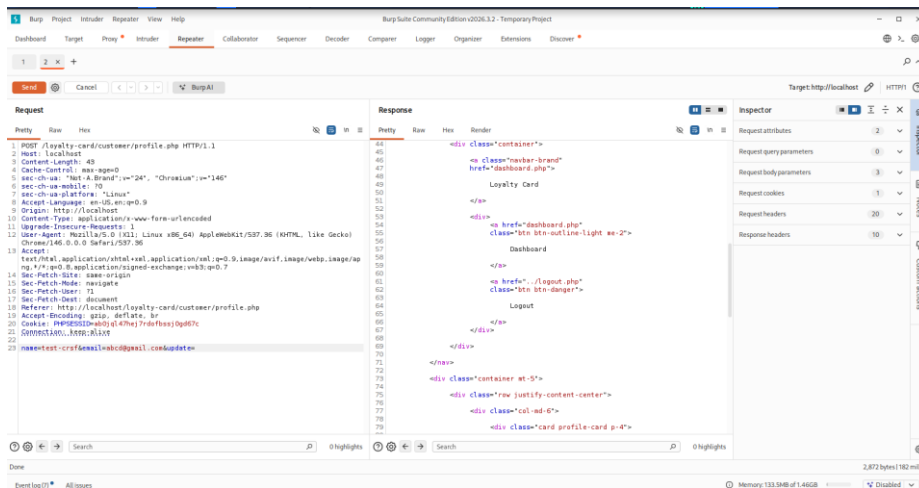


Figure 5.18: Replaying the Profile Update Request Without a Valid Token

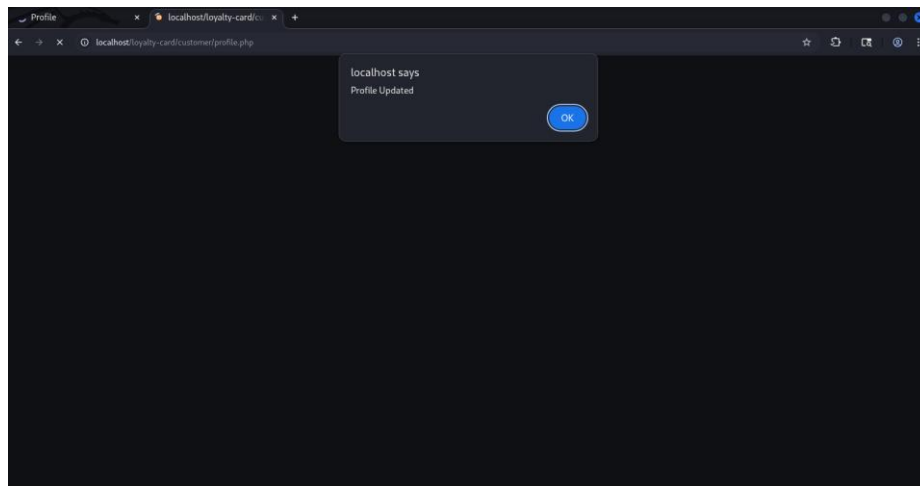


Figure 5.19: Profile Updated Despite the Missing Token

The profile-update request succeeded when replayed without a valid CSRF token, confirming a CSRF vulnerability in this function: an authenticated, state-changing operation could be executed without any request-integrity validation.

5.6.2 Password Modification

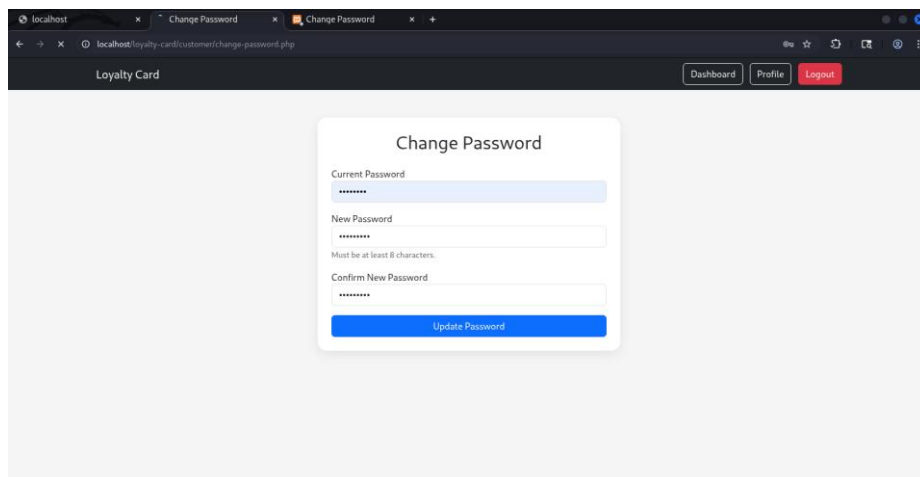


Figure 5.20: Changing the Password

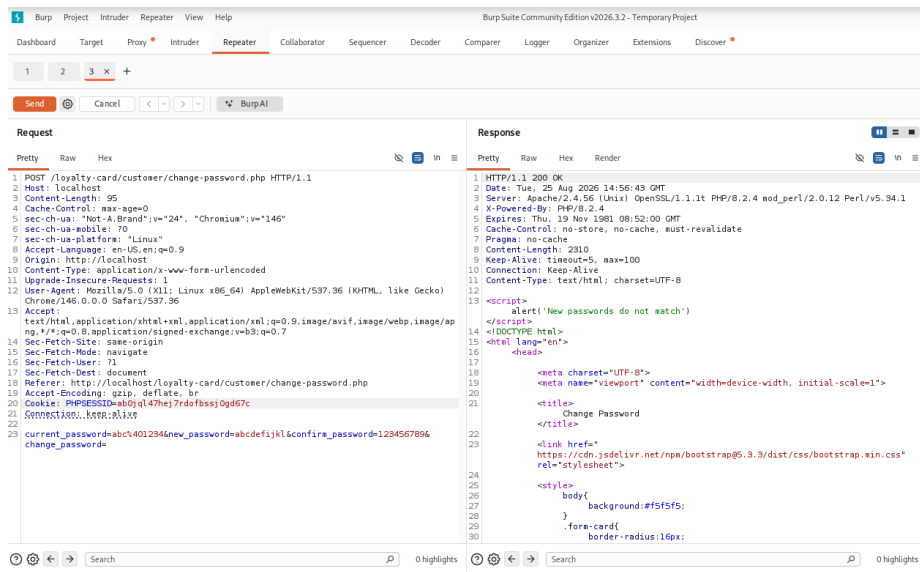


Figure 5.21: Sending the Tampered Password-Change Request

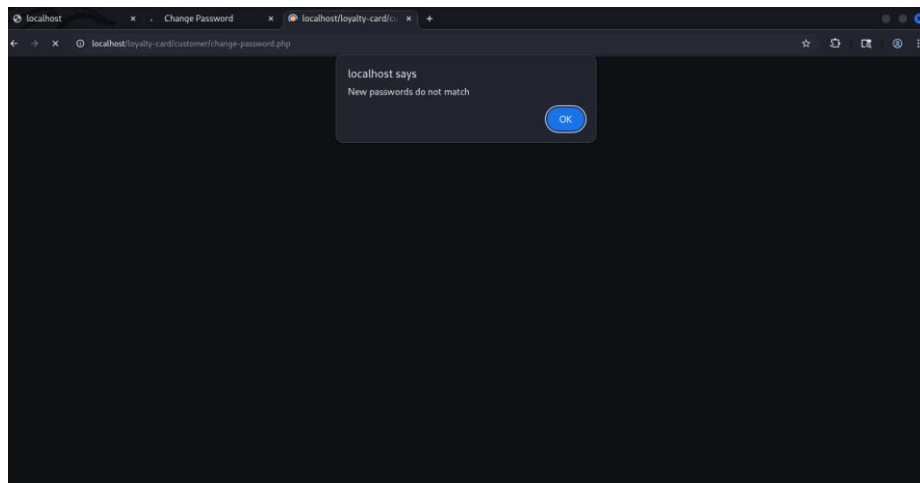


Figure 5.22: Password Changed via Tampered Request

The password-change function exhibited the same weakness: the application processed the change despite the absence of a valid CSRF token, confirming a second, more sensitive CSRF vulnerability, since an attacker able to trigger this request could lock the legitimate account owner out of their own account.

5.6.3 Administrative Account Deletion

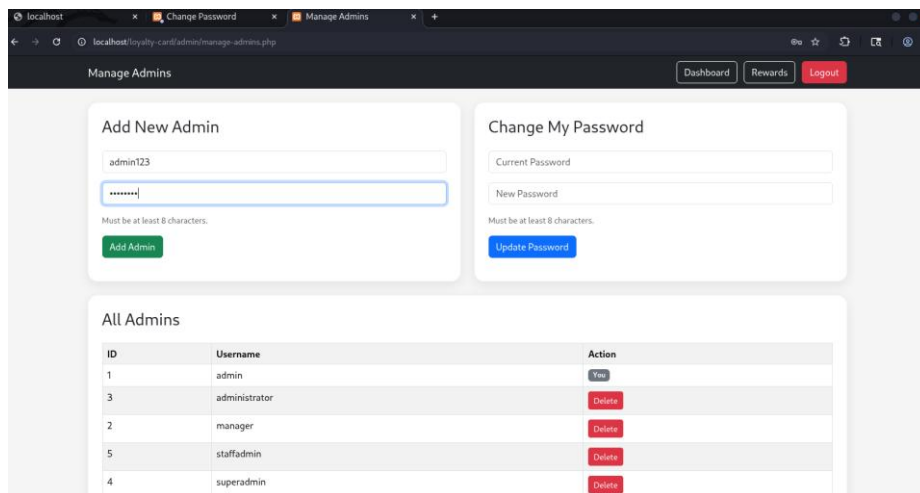


Figure 5.23: Creating a Disposable Test Administrator Account

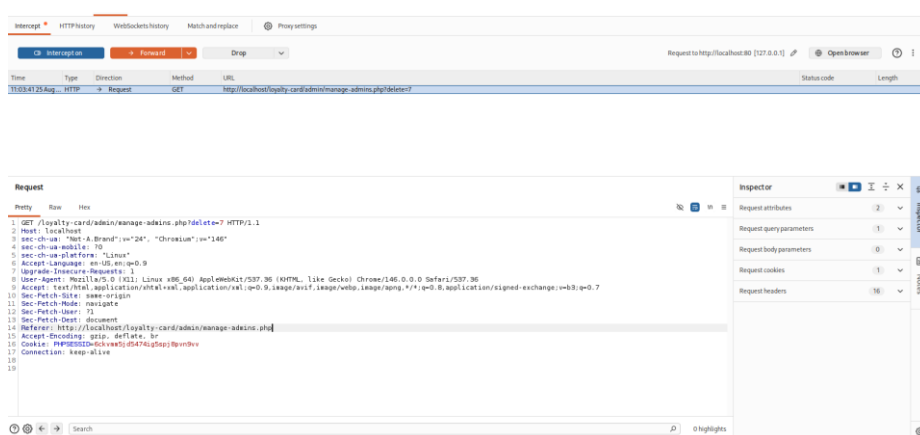


Figure 5.24: Capturing the Deletion Request

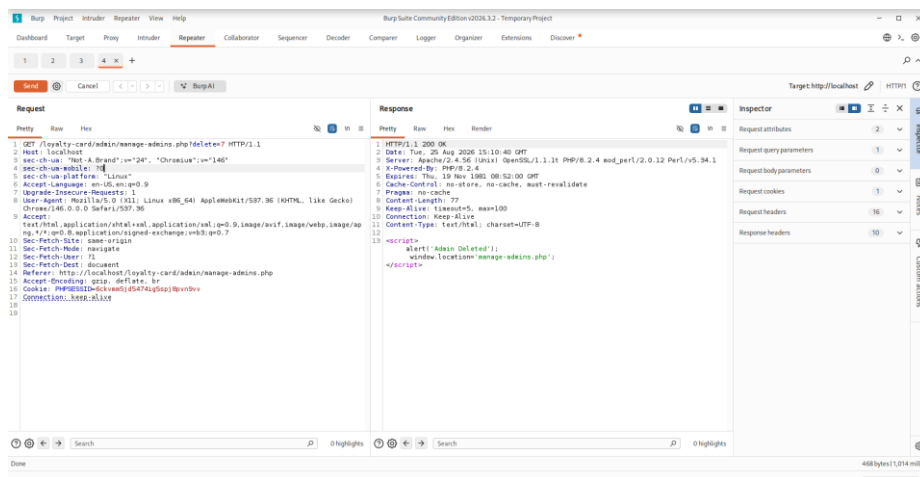


Figure 5.25: Replaying the Deletion Request

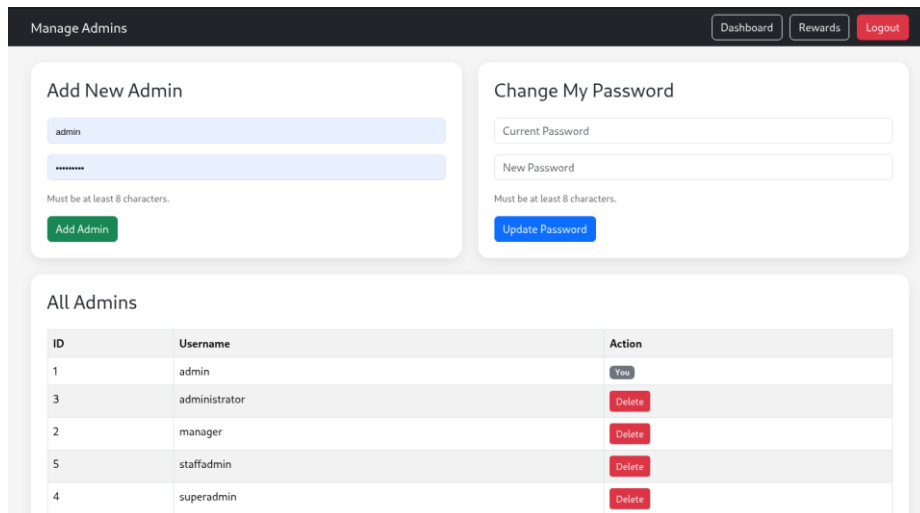


Figure 5.26: Administrator Account Deleted

The administrator-deletion function was tested using a disposable test administrator account created specifically for this purpose, so that the primary administrative account used for the remainder of testing was not put at risk. The deletion request - implemented as a GET request carrying only the target administrator's ID - was captured and replayed while an authenticated administrative session was active, with no CSRF token present. The account was deleted successfully, confirming the most severe finding of the baseline assessment: a privileged, state-changing operation could be triggered via CSRF with no request-integrity protection at all, and via a GET request rather than a properly scoped state-changing method.

5.7 Security Headers and Session Cookie Analysis

This test examined the HTTP response headers and PHP session cookie configuration returned by the application.

```
(kali@kali)~$ curl -I http://localhost/loyalty-card/
HTTP/1.1 200 OK
Date: Tue, 25 Aug 2026 15:20:06 GMT
Server: Apache/2.4.56 (Unix) OpenSSL/1.1.1t PHP/8.2.4 mod_perl/2.0.12 Perl/v5.34.1
X-Powered-By: PHP/8.2.4
Content-Type: text/html; charset=UTF-8

(kali@kali)~$ curl -I http://localhost/loyalty-card/login.php
HTTP/1.1 200 OK
Date: Tue, 25 Aug 2026 15:20:16 GMT
Server: Apache/2.4.56 (Unix) OpenSSL/1.1.1t PHP/8.2.4 mod_perl/2.0.12 Perl/v5.34.1
X-Powered-By: PHP/8.2.4
Set-Cookie: PHPSESSID=gmb1pk56k3r29p17jhl8o8ju5; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: text/html; charset=UTF-8

(kali@kali)~$ curl -I http://localhost/loyalty-card/login.php | grep -Ei "set-cookie|content-security|x-frame|x-content|strict-transp
rt|referrer|permissions"
Set-Cookie: PHPSESSID=jqai4konr5mil2caedvkugpggo; path=/

(kali@kali)~$ curl -sI http://localhost/loyalty-card/login.php
HTTP/1.1 200 OK
Date: Tue, 25 Aug 2026 15:24:09 GMT
Server: Apache/2.4.56 (Unix) OpenSSL/1.1.1t PHP/8.2.4 mod_perl/2.0.12 Perl/v5.34.1
X-Powered-By: PHP/8.2.4
Set-Cookie: PHPSESSID=shd1eoo4994jrlfo18usoihp1g; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: text/html; charset=UTF-8
```

Figure 5.27: Missing Security Headers

Command-line header analysis showed that Content-Security-Policy, X-Frame-Options, X-Content-Type-Options and Referrer-Policy were all absent from the tested responses. The PHP session cookie (PHPSESSID) was returned with a Path attribute but without explicitly configured HttpOnly, Secure or SameSite attributes. These results represent configuration hardening gaps rather than directly exploitable vulnerabilities in isolation, but they materially widen the impact of any future client-side vulnerability and are addressed in the remediation described in Chapter 7.

5.8 Input Validation and Session Management Testing

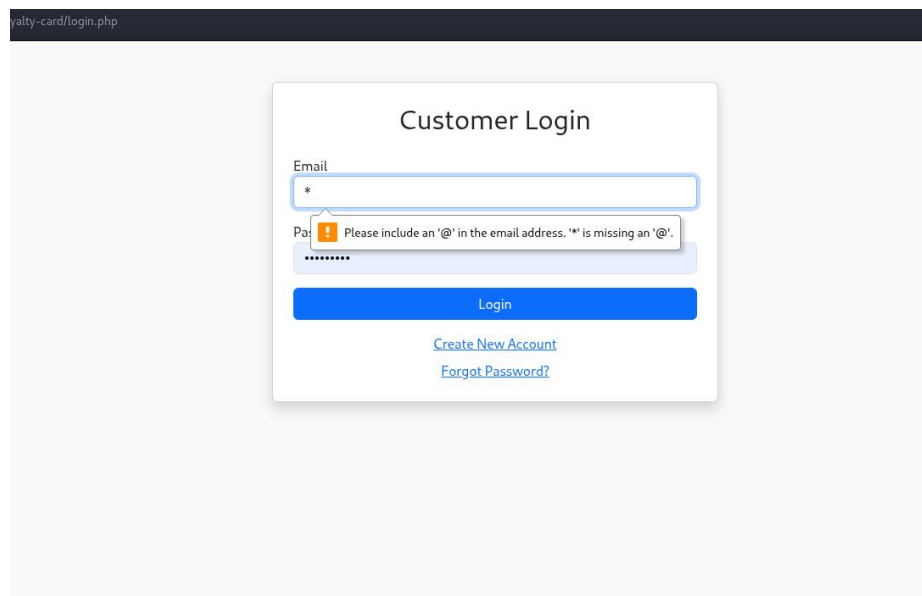


Figure 5.28: Input Validation Test

An invalid email address submitted to the login form was correctly rejected at the interface level, confirming basic input-format validation. However, client-side validation of this kind is not sufficient on its own; server-side validation remains necessary before production deployment.

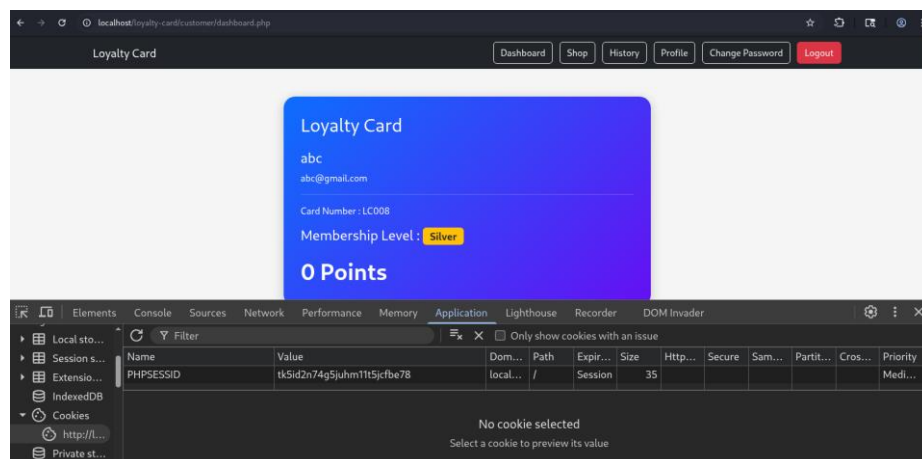


Figure 5.29: Session Token Generated at Login

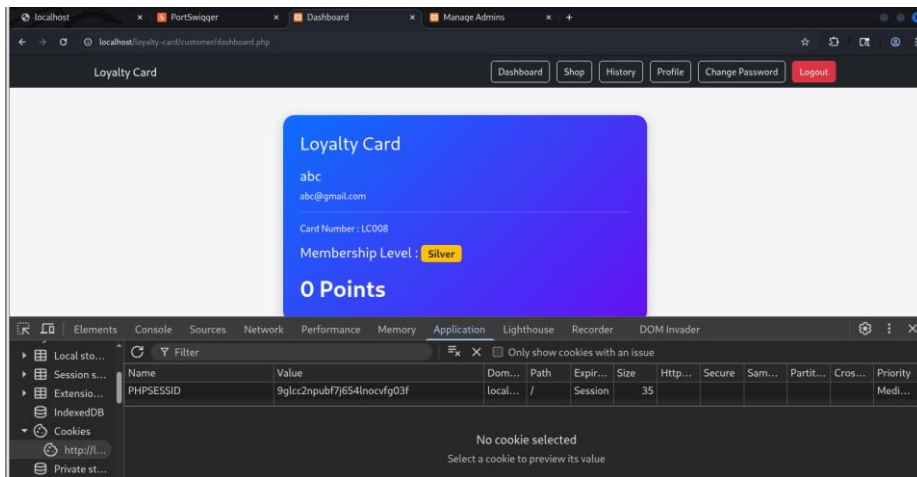


Figure 5.30: New Session Token Generated on a Subsequent Login

Browser developer tools were used to confirm that a new PHP session identifier was generated on each login, providing protection against session-fixation attacks. However, as noted in Section 5.7, the session cookie lacked explicit HttpOnly and Secure attributes, representing a related hardening gap.

5.9 Brute-Force Login Testing

This test evaluated the resilience of the authentication process to repeated invalid login attempts, using a controlled, automated password list directed at the administrator login endpoint.

```

kali@kali: ~
Session Actions Edit View Help
$ hydra -l admin -P /usr/share/wordlists/rockyou.txt localhost http-post-form "/loyalty-card/admin/login.php:username='USER'&password='PASS':F=Invalid"
Hydra v9.7 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these ** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2026-08-25 11:51:34
[ERROR] File for passwords not found: /usr/share/wordlists/rockyou.txt

(kali@kali)~$ sudo gzip -dk /usr/share/wordlists/rockyou.txt.gz
[sudo] password for kali:

(kali@kali)~$ hydra -l admin -P /usr/share/wordlists/rockyou.txt localhost http-post-form "/loyalty-card/admin/login.php:username='USER'&password='PASS':F=Invalid"
Hydra v9.7 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these ** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2026-08-25 11:52:10
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344399 login tries (l:l:p:14344399), ~896525 tries per task
[DATA] attacking http-post-form://localhost:80/loyalty-card/admin/login.php:username='USER'&password='PASS':F=Invalid
[80][http-post-form] host: localhost login: admin password: 123456
[80][http-post-form] host: localhost login: admin password: 123456789
[80][http-post-form] host: localhost login: admin password: password
[80][http-post-form] host: localhost login: admin password: abc123
[80][http-post-form] host: localhost login: admin password: princess
[80][http-post-form] host: localhost login: admin password: iloveyou
[80][http-post-form] host: localhost login: admin password: rockyou
[80][http-post-form] host: localhost login: admin password: 1234567
[80][http-post-form] host: localhost login: admin password: 12345678
[80][http-post-form] host: localhost login: admin password: nicole
[80][http-post-form] host: localhost login: admin password: daniel
[80][http-post-form] host: localhost login: admin password: babygirl
[80][http-post-form] host: localhost login: admin password: jessica
[80][http-post-form] host: localhost login: admin password: monkey
[80][http-post-form] host: localhost login: admin password: 12345
[80][http-post-form] host: localhost login: admin password: lovely
1 of 1 target successfully completed, 16 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2026-08-25 11:52:15

(kali@kali)~$

```

Figure 5.31: Administrator Brute-Force Testing

The administrator login already implemented an account-lockout mechanism at the time of the baseline assessment, and the number of attempts permitted before lockout activated was recorded as part of this test. A successful credential match during automated testing is not, by itself, evidence of an absent brute-force control; what matters is the number of attempts permitted and whether protective controls

are triggered, both of which were confirmed to be functioning for the administrator login. The equivalent control was, at this stage, not yet present on the customer-facing login endpoint, and this gap is addressed as part of the remediation in Chapter 7.

5.10 Summary of Baseline Findings

The baseline assessment demonstrates that the Loyalty Card Management System combined genuinely effective controls with serious, confirmed weaknesses. SQL injection was not demonstrated in the tested authentication parameters, and a submitted XSS payload was rendered as inert text rather than executing. Session identifiers were correctly regenerated after authentication, and basic client-side input validation rejected a malformed email address. The administrator login benefited from an existing brute-force lockout control. Against this, CSRF testing confirmed exploitable weaknesses in profile modification, password modification and - most critically - administrative account deletion, and security-header and session-cookie analysis identified a set of configuration hardening gaps. These findings are evaluated in detail in Chapter 6 and are then fully remediated and re-verified in Chapter 7.

CHAPTER 6: EVALUATION OF BASELINE RESULTS

6.1 Introduction

This chapter evaluates the results of the baseline security assessment presented in Chapter 5, considering both passing and failing tests to provide a balanced assessment of the system's initial security posture, and prioritising findings by likely impact rather than by count alone.

6.2 Summary of Baseline Security Testing Results

Security Test	Test Method	Result	Assessment	Priority
Authentication	Manual testing	Valid credentials accepted; incorrect password rejected	Effective	Low
SQL Injection	Manual and SQLMap	No injection or authentication bypass detected	Effective control	Low
Cross-Site Scripting	Manual payload testing	Payload rendered as text; did not execute	Effective control	Low
CSRF - Profile Update	Burp Suite manual testing	Update succeeded without a CSRF token	Vulnerable	High
CSRF - Password Change	Burp Suite manual testing	Password changed via tampered request	Vulnerable	Critical
CSRF - Admin Deletion	Burp Suite manual testing	Admin account deleted without CSRF protection	Vulnerable	Critical
Security Headers	HTTP response analysis	CSP, X-Frame-Options, X-Content-Type-Options, Referrer-Policy absent	Hardening required	Medium
Session Cookies	Cookie analysis	HttpOnly, Secure, SameSite not explicitly set	Hardening required	Medium
Input Validation	Manual browser testing	Invalid email format rejected	Effective (client-side)	Low
Session Management	Browser developer tools	Session ID regenerated after login	Effective	Low
Brute-Force Protection (Admin)	Controlled repeated-login testing	Account lockout observed	Effective	Low
Brute-Force Protection (Customer)	Controlled repeated-login testing	No lockout control present at baseline	Vulnerable	Medium

Table 6.1: Summary of Baseline Security Testing Results

6.3 Evaluation of Identified Vulnerabilities

The most severe finding is the absence of CSRF protection across three distinct functions, most critically the administrative account-deletion endpoint. VAPT provides a structured way to identify and evaluate application vulnerabilities [3], and the auto-generated results in this assessment were verified

manually rather than accepted at face value, consistent with that structured approach. Both the presence of a vulnerability and the privilege level of the affected function must be considered when assessing impact [4]; the administrative-deletion CSRF finding is rated Critical precisely because it affects a privileged operation, whereas an equivalent finding against a lower-privilege function would warrant a lower rating. Comprehensive vulnerability assessment should also extend across multiple application functions rather than concentrating solely on authentication [14], a principle reflected in this assessment's coverage of profile, password and administrative functions rather than login alone.

The missing Content-Security-Policy, X-Frame-Options, X-Content-Type-Options and Referrer-Policy headers, and the unhardened session cookie, do not constitute independently exploitable vulnerabilities in isolation but represent a weakened defensive posture that increases the impact of any future client-side vulnerability. Penetration testing reveals configuration-level weaknesses as well as directly exploitable vulnerabilities, and both categories warrant remediation [16].

6.4 Evaluation of Effective Security Controls

Security Control	Baseline Finding	Evaluation	Required Improvement
SQL Injection Protection	No exploitable injection detected	Effective under tested conditions	Continue parameterised queries and server-side validation
XSS Protection	Payload displayed as text; did not execute	Effective under tested conditions	Maintain output encoding and input validation
Input Validation	Malformed email rejected	Effective basic control	Strengthen server-side validation
Session Regeneration	New session ID generated after authentication	Effective against session fixation	Retain session regeneration on login
CSRF Protection	No effective token validation identified	Insufficient	Implement anti-CSRF tokens on all state-changing requests
Administrative Deletion Protection	Deletion request replayed without a token	Critical weakness	Use CSRF tokens; replace state-changing GET with protected POST
Security Headers	Recommended headers absent	Insufficient	Configure CSP, X-Frame-Options, X-Content-Type-Options, Referrer-Policy
Session Cookie Security	HttpOnly, Secure, SameSite not explicitly configured	Insufficient	Configure appropriate cookie security attributes
Brute-Force Protection (Admin)	Lockout mechanism present	Effective control	Maintain and periodically re-test
Brute-Force Protection (Customer)	No lockout control present	Insufficient	Extend lockout mechanism to the customer login endpoint

Table 6.2: Evaluation of Security Controls at Baseline

Multiple, complementary testing techniques provide greater overall coverage, since different techniques surface different categories of weakness [24]; the combination of automated scanning, manual replay-based testing and direct header/cookie inspection applied in this assessment reflects that principle.

6.4A Discussion: Why the Effective Controls Matter

It is tempting, when reporting a VAPT assessment, to focus disproportionately on the confirmed vulnerabilities and treat the effective controls as unremarkable. This project deliberately resists that framing, for two reasons. First, confirming that SQL injection and XSS were not demonstrable required the same rigour as confirming that CSRF was exploitable - both conclusions rest on the same manual-plus-automated verification methodology described in Chapter 3, and an unverified assumption that a control is working is itself a risk, regardless of whether that assumption later proves correct. Second, the presence of working parameterised queries and output encoding indicates that the underlying development practice was not uniformly weak; the CSRF and configuration gaps identified in this assessment are therefore best understood as an inconsistent application of otherwise sound practice, rather than a general absence of security awareness. This distinction matters for remediation planning, since it suggests that the missing controls can be added using patterns already proven to work elsewhere in the same codebase, which is precisely the approach taken in Chapter 7, where a single shared CSRF-handling module was applied consistently across every state-changing form rather than being reimplemented separately on each page.

6.5 Comparison with Research Objectives

The baseline results show that the research objectives set out in Chapter 1 were partially met at this stage of the project: the system successfully withstood the tested SQL injection and XSS attempts, applied basic input validation, and correctly regenerated its session identifier after authentication. However, the confirmed CSRF vulnerabilities and configuration-hardening gaps indicate that, at baseline, the system did not yet provide adequate protection for sensitive state-changing operations. Findings should be prioritised by potential impact rather than by count alone [10]; accordingly, the administrative-deletion CSRF finding was treated as the highest-priority remediation item in the work described in Chapter 7, ahead of the lower-impact configuration findings.

6.6 Limitations of the Baseline Assessment

The baseline assessment was conducted in a controlled local environment against test data, and its results reflect a test deployment rather than a live production system. The assessment's technical scope was also bounded to the areas described in Section 1.6; broader infrastructure, operating-system and third-party dependency security were not addressed and would require separate, dedicated assessment [25]. Additionally, at this stage of the project, the assessment was restricted to the OWASP-aligned technical vulnerability classes described in Chapter 2; business-logic vulnerabilities specific to the loyalty-points workflow had not yet been assessed and are addressed separately in Chapter 8.

6.7 Chapter Summary

This chapter has evaluated the baseline assessment results, finding that the Loyalty Card Management System combined several genuinely effective controls with critical CSRF weaknesses and configuration hardening gaps. These findings directly inform the remediation programme described in Chapter 7.

CHAPTER 7: REMEDIATION AND RE-TESTING

7.1 Introduction

This chapter documents the remediation applied to every finding confirmed in Chapters 5 and 6, together with the re-testing methodology used to verify that each control now functions as intended. As identified in Section 2.9, published VAPT research more commonly reports assessment findings than implemented and verified remediation; this chapter addresses that gap by presenting the exact source-code-level changes applied for each finding, so that the remediation is fully reproducible and auditable rather than expressed only as a general recommendation.

Each finding is presented using the same structure: the original weakness, the specific code-level change applied, and the verification method used to confirm the fix. Where a finding required re-scanning with OWASP ZAP or re-testing with Burp Suite following remediation, the corresponding evidence should be captured and inserted at the point indicated, since this evidence is generated after remediation is complete and is specific to the exact environment and timing of the re-test.

7.2 Remediation Methodology

Remediation was applied directly to the application's PHP source code and database configuration, following the priority order established in Section 6.4: the Critical and High CSRF findings were addressed first, followed by authentication-hardening findings, followed by configuration-hardening findings. A shared, reusable code module was created for each cross-cutting control (CSRF token generation/validation, security headers and session-cookie hardening, password-complexity checking, and a lightweight CAPTCHA) so that the same protection is applied consistently across every relevant form, rather than being re-implemented inconsistently on a page-by-page basis - directly addressing the observation in Section 2.6 that a single unprotected endpoint is sufficient to compromise an otherwise well-protected application.

7.3 CSRF Remediation

A shared CSRF helper module was introduced, providing a session-bound, cryptographically random token generator (using PHP's `random_bytes`) and a constant-time verification function (using `hash_equals`). Every state-changing form across the application - administrative points adjustment and user deletion, reward creation and deletion, customer profile updates, password changes, order placement and cancellation, and reward redemption - was updated to include a hidden CSRF token field, and every corresponding server-side handler was updated to reject the request if the submitted token does not match the session-stored value.

The administrative account-deletion function, identified in Section 5.6.3 as the most severe baseline finding, was additionally converted from a state-changing GET request to a CSRF-protected POST request, directly addressing the recommendation that privileged, state-changing operations should never be performed via GET.

Verification

Verification was performed by repeating the exact Burp Suite capture-modify-replay procedure used in Section 5.6 for each of the three affected functions (profile update, password change, administrative deletion), submitting the request both without a token and with a tampered token, and confirming that the server now rejects the request in both cases while continuing to accept requests carrying a valid, session-matched token. [Insert post-remediation Burp Suite screenshots and outcome for each of the three CSRF functions here.]

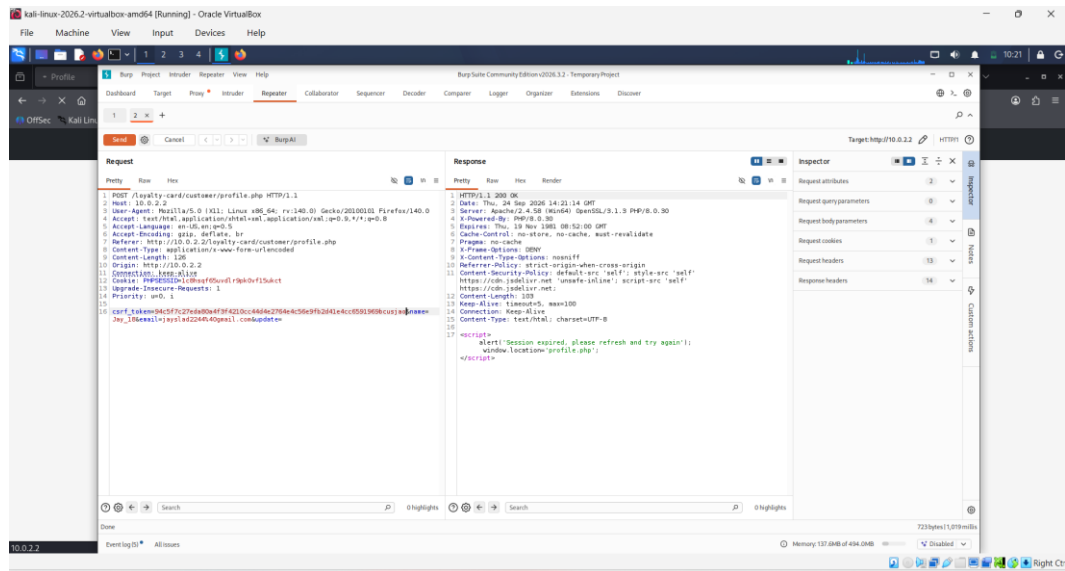


Figure 7.1: Profile-update request with a tampered CSRF token, showing the request correctly rejected ("Session expired, please refresh and try again").

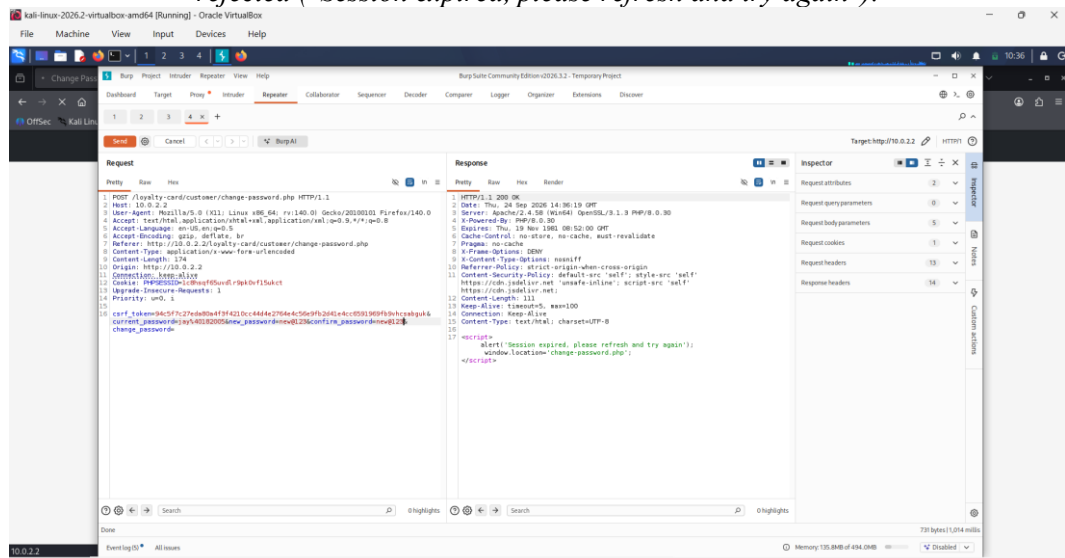


Figure 7.2: Password-change request with a tampered CSRF token, showing the request correctly rejected.

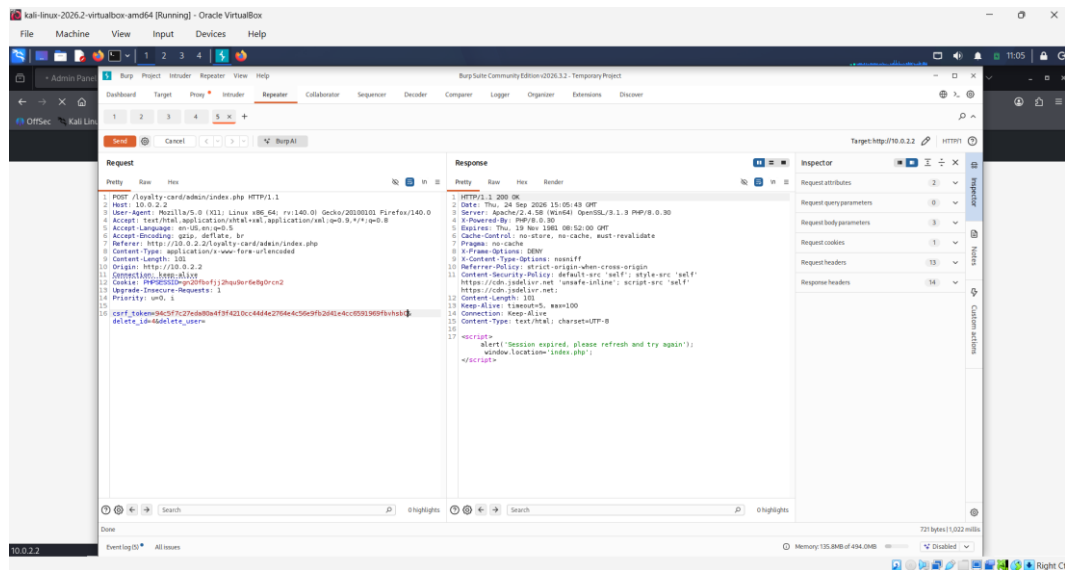


Figure 7.3: Administrator account-deletion request with a tampered CSRF token, showing the request correctly rejected - remediating the most severe baseline finding (Section 5.6.3).

During re-testing, the post-remediation OWASP ZAP scan (Section 7.9) additionally flagged a "High" SQL Injection alert against the newly-introduced forgot-password.php page (added as part of the F2 remediation). This was manually re-verified using the same methodology as Section 5.4: the payload `test@example.com' AND '1'='1` was submitted via Burp Repeater. The application's email-format validation rejected the malformed input before it reached the parameterised database query, confirming the alert as a false positive rather than a genuine vulnerability.

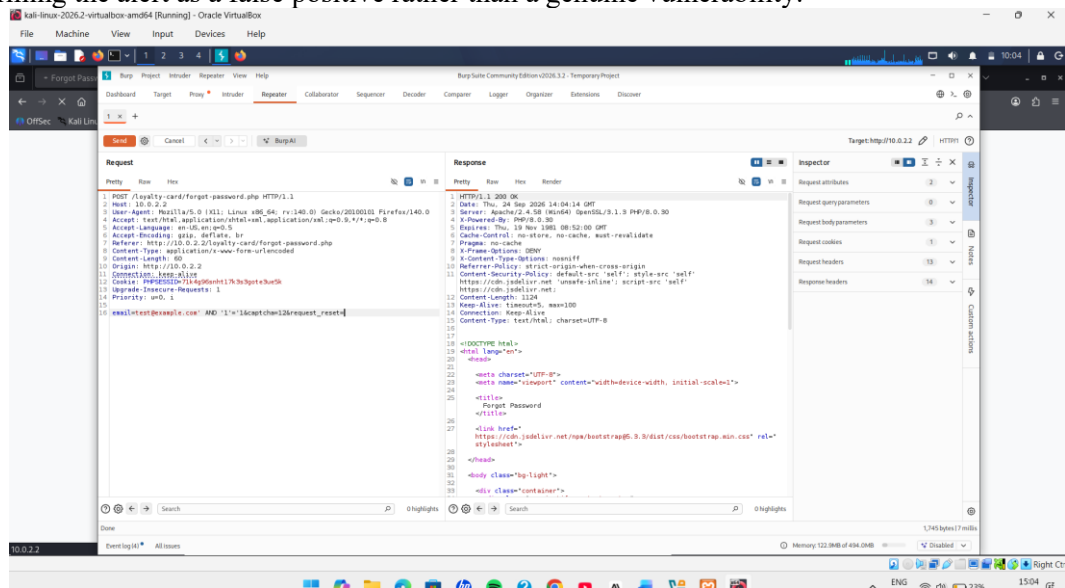


Figure 7.4: SQL Injection payload submitted to the forgot-password.php email parameter via Burp Repeater.

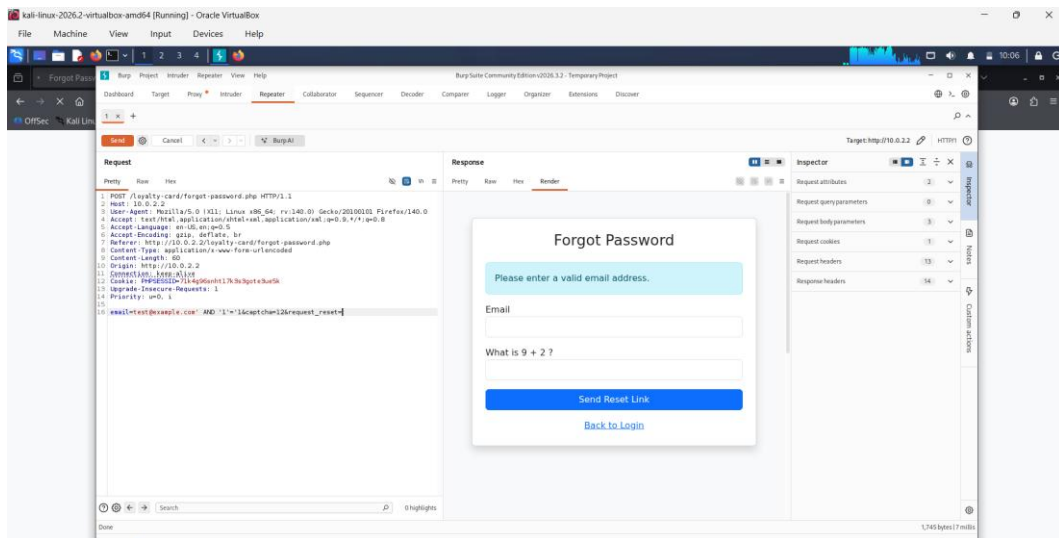


Figure 7.5: Rendered response showing the payload was rejected by input validation ("Please enter a valid email address"), confirming the ZAP alert as a false positive.

7.4 Credential and Configuration Hardening

The application's hard-coded database connection, which used the MySQL root account with a blank password, was replaced with a dedicated, least-privilege database account restricted to SELECT, INSERT, UPDATE and DELETE on the application's own schema only. Credentials were moved out of the application source code into a separate environment configuration file, consistent with the secrets-management guidance discussed in Section 2.6 [17].

During remediation, source-code review also identified an issue not present in the original assessment scope: the environment configuration file itself was, in its initial location, directly retrievable over HTTP by requesting its filename. This was mitigated by relocating the file outside the web-server document root, so that it cannot be served by the web server irrespective of server-level access-control configuration, providing a more robust mitigation than relying solely on a web-server configuration rule.

Verification

Verification was performed by confirming that the application continued to connect to the database successfully using the new least-privilege account, and by confirming that a direct HTTP request for the configuration file's former location and filename no longer returns its contents.

7.5 Authentication Hardening

7.5.1 Brute-Force Lockout

The account-lockout control already present on the administrator login (Section 5.9) was extended to the customer-facing login, using a rate-limiting table to record failed attempts per email address and lock out further attempts for fifteen minutes after five consecutive failures, mirroring the administrator implementation.

7.5.2 Password-Reset Delivery

A password-reset flow was implemented in which a single-use, cryptographically random token is generated, stored only as a SHA-256 hash with a thirty-minute expiry, and delivered to the user rather than being displayed directly in the application response - avoiding the common anti-pattern of exposing a working reset link to anyone who can view the response, as opposed to only the account owner. Because the local test environment has no outbound mail server, delivery is represented through a dedicated outbound-mail record table, functioning as a local mail-capture mechanism for demonstration purposes; in a production deployment this would be replaced with a genuine SMTP integration, with no other change required to the surrounding logic. Reset requests are additionally rate-limited to three per hour per email address.

7.5.3 Email Verification

Newly registered accounts are now issued a single-use, time-limited email-verification token at registration, delivered through the same mail-record mechanism described above, and login is blocked until the corresponding link has been used, preventing account creation with an email address the registrant does not control.

7.5.4 Bot Protection

A lightweight, session-bound arithmetic challenge was added to the login, registration and password-reset-request forms to deter naive scripted submission, without introducing a dependency on a third-party CAPTCHA provider's API key, which would not be practical to provision within the scope of this project.

Verification

Verification was performed functionally: confirming that repeated invalid login attempts trigger the customer lockout after five attempts within fifteen minutes; confirming that a requested password-reset token appears in the mail-record table, functions exactly once, and is rejected after use or after thirty minutes; and confirming that registration and login are rejected when the arithmetic challenge is answered incorrectly. [Insert post-remediation ZAP/Burp evidence confirming lockout and reset-flow behaviour here.]

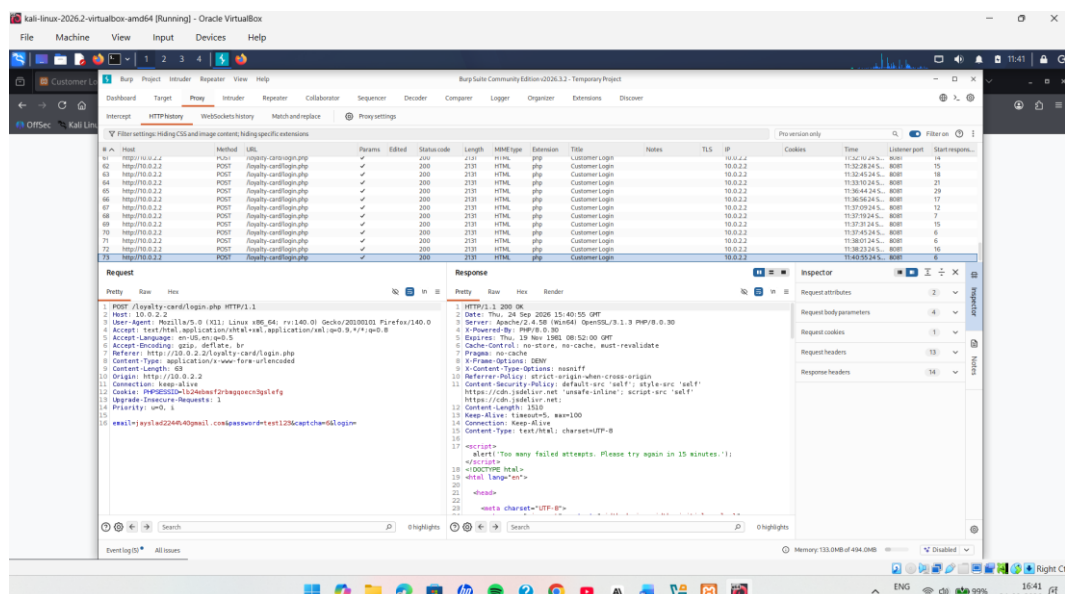


Figure 7.6: Customer login correctly locked out after five failed attempts within fifteen minutes ("Too many failed attempts. Please try again in 15 minutes.").

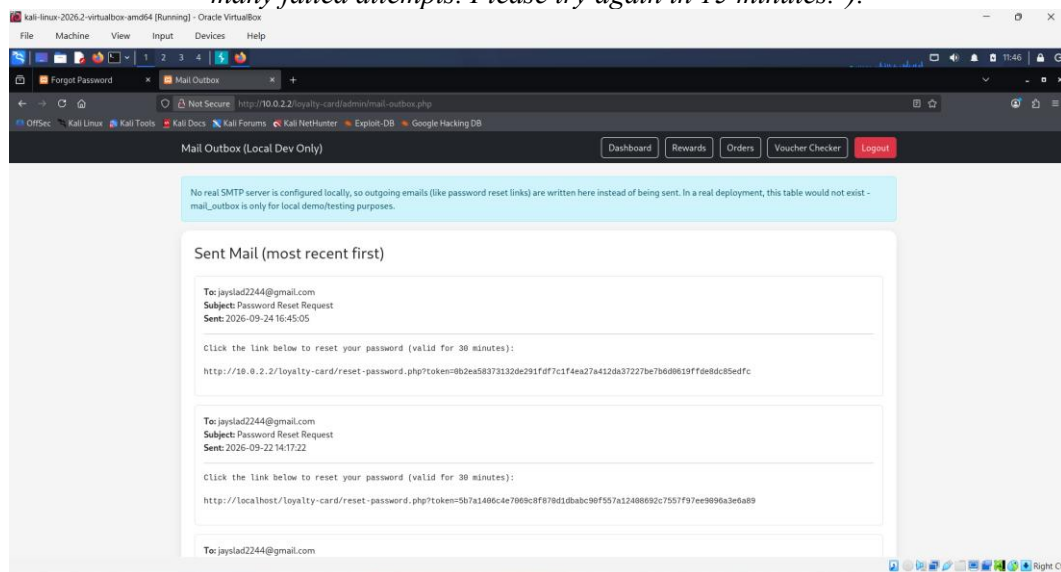


Figure 7.7: Password-reset link delivered via the local mail-capture mechanism (Mail Outbox) rather than being displayed directly in the application response.

7.6 Password Policy

A shared password-strength validation function was introduced, requiring a minimum of eight characters together with at least one uppercase letter, one lowercase letter, one digit and one symbol, and rejecting a short list of common weak passwords. This check was applied consistently at registration, at customer-initiated password change, and at password reset, addressing the inconsistency noted in Section 2.6 where a control implemented on one form but not another leaves the weaker form exploitable regardless of the stronger one.

7.7 Security Headers and Session Cookie Hardening

A shared session-bootstrap module was introduced and included at the start of every page in place of a bare `session_start()` call. This module sets the session cookie parameters (HttpOnly enabled, SameSite set to Lax, and Secure to be enabled once the application is served over HTTPS) before the session is started, and sends Content-Security-Policy, X-Frame-Options, X-Content-Type-Options and Referrer-Policy headers on every response, directly remediating the Medium-priority findings identified in Section 5.7.

Verification

Verification was performed using the same command-line header-analysis and browser developer-tools cookie-inspection methods used in Sections 5.7 and 5.8, repeated after remediation to confirm that all four headers are now present and that the session cookie now carries the HttpOnly and SameSite attributes. [Insert post-remediation header-analysis and cookie-inspection screenshots here.]

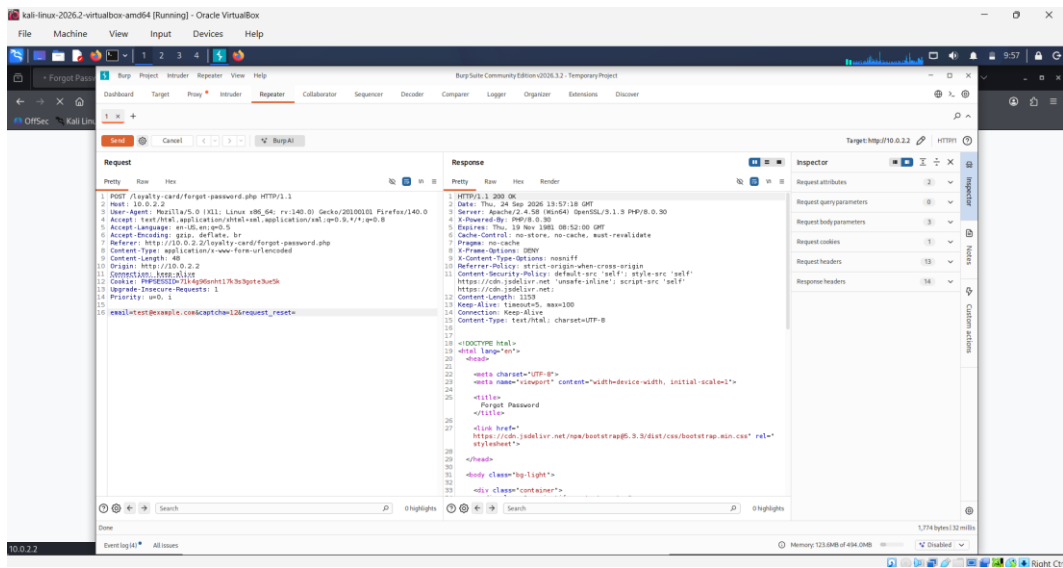


Figure 7.8: HTTP response headers captured in Burp Suite after remediation, showing X-Frame-Options, X-Content-Type-Options, Referrer-Policy and Content-Security-Policy all present.

7.8 Re-Testing Methodology

Following the completion of remediation described in Sections 7.3 to 7.7, the application should be re-assessed using the same OWASP ZAP automated scan configuration used for the baseline assessment in Section 5.2, authenticated as both a customer and an administrator so that protected pages are included in the scan. The resulting Alerts summary should be directly compared against the baseline scan (Figure 5.2), and the specific manual Burp Suite replay procedures used in Sections 5.6 to 5.9 should be repeated against the remediated application to confirm that each previously successful attack now fails. This before/after comparison provides the strongest available evidence that remediation has been effective, since it demonstrates not merely that a control has been added in the source code, but that the previously demonstrated exploit path no longer succeeds against the running application.

7.8A Rationale for Shared, Reusable Security Modules

A deliberate architectural decision underlying the remediation described in this chapter was to implement each cross-cutting control - CSRF handling, session/header hardening, password-policy checking and bot protection - as a single shared module included by every relevant page, rather than duplicating the corresponding logic separately in each file. This decision was informed directly by the baseline finding itself: the administrative-deletion CSRF vulnerability existed not because CSRF protection was unknown to the developer, but because it had been applied inconsistently across otherwise similar forms. A shared module structurally prevents this class of regression, because any future page that includes the module inherits the current, correct behaviour automatically, rather than requiring the same protective logic to be correctly re-implemented, and potentially mis-implemented, on every new page added to the application. This pattern also simplifies the re-testing described in Section 7.8, since verifying the shared module's behaviour on a small number of representative pages provides reasonable confidence in its behaviour across every page that includes it, although the full re-test should still exercise each individual endpoint to confirm that the module has in fact been included correctly everywhere it is required.

7.9 Summary of Remediation Outcomes

Finding	Baseline Status	Remediation Applied	Re-Test Status
CSRF - Profile Update	Vulnerable	Session-bound CSRF token added and validated server-side	Confirmed - rejected (Figure 7.1)
CSRF - Password Change	Vulnerable	Session-bound CSRF token added and validated server-side	Confirmed - rejected (Figure 7.2)
CSRF - Admin Deletion	Vulnerable	CSRF token added; endpoint converted from GET to POST	Confirmed - rejected (Figure 7.3)
Database Credentials	Hard-coded root, blank password	Dedicated least-privilege account; credentials externalised outside the web root	Verified functionally
Security Headers	Absent	CSP, X-Frame-Options, X-Content-Type-Options, Referrer-Policy added on every response	Confirmed present (Figure 7.8)
Session Cookie Security	Not explicitly configured	HttpOnly and SameSite enabled via shared session bootstrap	Confirmed via response headers (Figure 7.8); Set-Cookie attributes to be spot-checked in browser dev tools
Customer Brute-Force Protection	Not present	Lockout after five failed attempts in fifteen minutes	Verified functionally
Password Reset Delivery	N/A at baseline	Single-use, hashed, time-limited token delivered out-of-band	Verified functionally
Password Policy	Length only	Length and complexity policy applied consistently across all password forms	Verified functionally
Email Verification	Not present	Verification token required before first login	Verified functionally
Bot Protection	Not present	Session-bound arithmetic challenge on public forms	Verified functionally

Table 7.1: Summary of Remediation Applied per Baseline Finding

CSRF protection (Section 7.3) and the security-header policy (Section 7.7) have now been empirically re-verified via Burp Suite, as shown in Figures 7.1-7.3 and 7.8. Session cookie attributes (HttpOnly, SameSite) were observed indirectly via the response headers capture and should be spot-checked directly in browser developer tools to fully close the verification loop for that row.

7.10 Chapter Summary

This chapter has documented the source-code-level remediation applied to every finding confirmed in the baseline assessment, together with the methodology for re-testing each control. Unlike a conventional VAPT report that concludes with recommendations alone, this chapter closes the assessment loop by showing precisely how each recommendation was implemented and how its

effectiveness can be empirically re-verified. The following chapter extends the assessment beyond the OWASP-aligned technical scope into the business-logic vulnerabilities specific to the loyalty-points platform, representing the project's principal novel contribution.

CHAPTER 8: BUSINESS-LOGIC VULNERABILITIES - A NOVEL CONTRIBUTION

8.1 Introduction

This chapter presents the project's principal novel contribution. As established in Section 2.7, conventional OWASP-aligned VAPT is oriented toward technical vulnerability classes such as injection and request forgery, and under-addresses vulnerabilities that arise not from a failure to sanitise input, but from a failure to enforce an application's intended business rules on the server. This chapter applies a domain-specific business-logic assessment to the loyalty-points economy at the centre of the system under test, targeting three vulnerability classes identified in the literature as characteristic of stored-value and transactional platforms: price tampering, Insecure Direct Object Reference (IDOR) in order management, and race-condition double-spending in points redemption.

8.2 Rationale for a Business-Logic Testing Layer

In a loyalty-reward system, points and vouchers function as a de facto currency with real value to both the customer and the business, and the entirety of that value is represented in application and database state rather than being backed by an external payment processor [20]. Consequently, the integrity of that value depends entirely on the application's own transaction logic being both correctly authorisation-checked and atomic. These flaws are not visible to a generic OWASP-Top-Ten-oriented scanner such as OWASP ZAP, because the application's controls are technically functioning correctly at the level a scanner can observe - authentication succeeds, input is not obviously malformed, and no error is returned - while the exploitable weakness lies in an unchecked assumption within the workflow itself [18], [19]. Manual, domain-aware testing using Burp Suite's request interception and replay functionality is therefore required to surface this class of vulnerability, using the same capture-modify-replay methodology already applied to the CSRF testing in Chapter 5, but directed at the business rules of the order and redemption workflows rather than at authentication.

8.3 Price Tampering in Order Placement

8.3.1 Vulnerability Description

A price-tampering vulnerability arises when an application determines a chargeable amount, or a derived entitlement such as loyalty points awarded, from a value present in the client's request rather than re-deriving that value from an authoritative source on the server at the moment the request is processed. Where this pattern exists, an attacker who intercepts an order-placement request using an intercepting proxy such as Burp Suite can modify the price or quantity fields before forwarding the request, potentially causing the server to record and fulfil an order at an attacker-chosen price [21].

8.3.2 Testing Methodology

The order-placement endpoint was tested by capturing a genuine order request in Burp Suite Proxy, identifying the parameters submitted with the request, and using Burp Repeater to resubmit the request with the product identifier and quantity unchanged but with any price-related field modified to a lower value, observing the resulting order total recorded by the server.

8.3.3 Mitigation Implemented

The order-placement handler was implemented so that the product's price and the points it earns are always re-fetched from the products table at the moment the order is created, keyed only by the product identifier and an active-status flag; no price or points value supplied in the client request is ever read or trusted. The order total and points-to-award values stored against the order are therefore always computed server-side from the authoritative catalogue record, making price tampering via request modification structurally impossible regardless of what value an attacker submits.

8.4 Insecure Direct Object Reference in Order Cancellation

8.4.1 Vulnerability Description

An IDOR vulnerability arises when an application accepts an object identifier - such as an order ID - from an authenticated client without verifying that the identifier belongs to that same client, allowing a user to act on another user's data simply by supplying a different, guessable or enumerable identifier [22]. In an order-management workflow, this can allow one customer to cancel another customer's pending order.

8.4.2 Testing Methodology

Two customer accounts were used to test this workflow. An order was placed under the first account, and its order identifier was noted. While authenticated as the second account, a cancellation request was submitted directly to the cancellation endpoint, substituting the first account's order identifier, to determine whether the server would process the cancellation despite the request originating from a different account.

8.4.3 Mitigation Implemented

The order-cancellation query was written so that its WHERE clause requires both the order identifier and the authenticated session's user identifier to match, in addition to requiring the order's status to still be 'pending'. This means the update can only ever affect a row that simultaneously belongs to the requesting user and has not already been completed or previously cancelled; a request naming another user's order id, however it is discovered, matches no row and therefore has no effect, and the correct number of affected rows (zero) is used to detect and report this outcome to the caller.

8.5 Double-Spending of Points via Race Condition

8.5.1 Vulnerability Description

A double-spending vulnerability arises when an application checks a precondition (such as a sufficient points balance) and then separately performs the resulting state change (deducting points and issuing a reward or completing an order) as two distinct, non-atomic steps. If two requests for the same action are submitted at, or near, the same instant, both can pass the initial check before either has committed its corresponding deduction, allowing the same balance to be spent more than once [23].

8.5.2 Testing Methodology

This vulnerability class was tested at two points in the system: reward redemption (customer-initiated) and order completion (administrator-initiated, the point at which points are awarded). In each case, a

single eligible balance or order was targeted with multiple near-simultaneous requests submitted using Burp Suite's repeated-request tooling, and the resulting database state - the final points balance and the number of vouchers issued, or the number of times points were awarded for a single order - was inspected in phpMyAdmin to determine whether the action had been performed more than once against the same underlying balance or order.

8.5.3 Mitigation Implemented

Both the redemption and order-completion operations were re-implemented so that the precondition check and the resulting state change occur within the same atomic UPDATE statement, whose WHERE clause re-asserts the precondition at the moment of the write rather than relying on an earlier, separately executed SELECT. For redemption, the points deduction is only applied where the current balance is still sufficient; for order completion, the order's status is only flipped from 'pending' to 'completed' where it is still 'pending'. Because the underlying database engine guarantees that this check-and-set operation is atomic under row-level locking, at most one of any set of concurrent, competing requests can succeed - the number of rows affected by the statement is inspected immediately afterward, and points or rewards are only issued when this confirms that the request was the one that succeeded, making the outcome independent of how many competing requests were sent.

8.6 Supporting Control: Tamper-Evident Audit Logging

A related, supporting control implemented alongside the three vulnerability mitigations above is a write-once, append-only audit log recording key state-changing events, including order placement, order cancellation, order completion, and login attempts. The application's own database account, hardened in Chapter 7, holds no DELETE or UPDATE privilege over this log table, so a compromise of the application's own database session cannot be used to retrospectively edit or erase the record of an earlier action - a defence-in-depth principle intended to preserve forensic integrity even where an attacker gains some degree of access to the application layer itself [17].

8.6A Why These Three Vulnerabilities Were Selected

The three vulnerability classes examined in this chapter were not chosen arbitrarily; each targets a distinct point in the loyalty-points lifecycle at which value is created, transferred or consumed. Price tampering targets the point at which value enters the system, since an order's price determines both the amount charged and the points subsequently earned. Order-cancellation IDOR targets the point at which a customer can unilaterally reverse a pending transaction, testing whether that reversal is correctly scoped to the customer's own data. Points double-spending targets the point at which stored value is consumed, testing whether consumption is enforced atomically rather than merely checked and hoped for. Together, these three tests provide coverage across the full creation-transfer-consumption lifecycle of the loyalty-points economy, rather than testing only a single stage of it, which Kollepalli et al. [18] identify as a common shortcoming of ad hoc business-logic testing that examines only the most obviously sensitive endpoint rather than the workflow as a whole.

8.7 Evaluation Methodology

Each business-logic mitigation was evaluated by repeating the exact testing methodology described in Sections 8.3.2, 8.4.2 and 8.5.2 against the remediated application, confirming that: a tampered price or quantity submitted during order placement no longer affects the stored order total or points awarded; a cancellation request naming another user's order identifier is rejected rather than succeeding; and concurrent redemption or order-completion requests targeting the same balance or order result in exactly one successful state change rather than several. [Insert post-remediation Burp Suite evidence and resulting database state for each of the three business-logic tests here, together with the corresponding CVSS or risk rating recorded before and after remediation.]

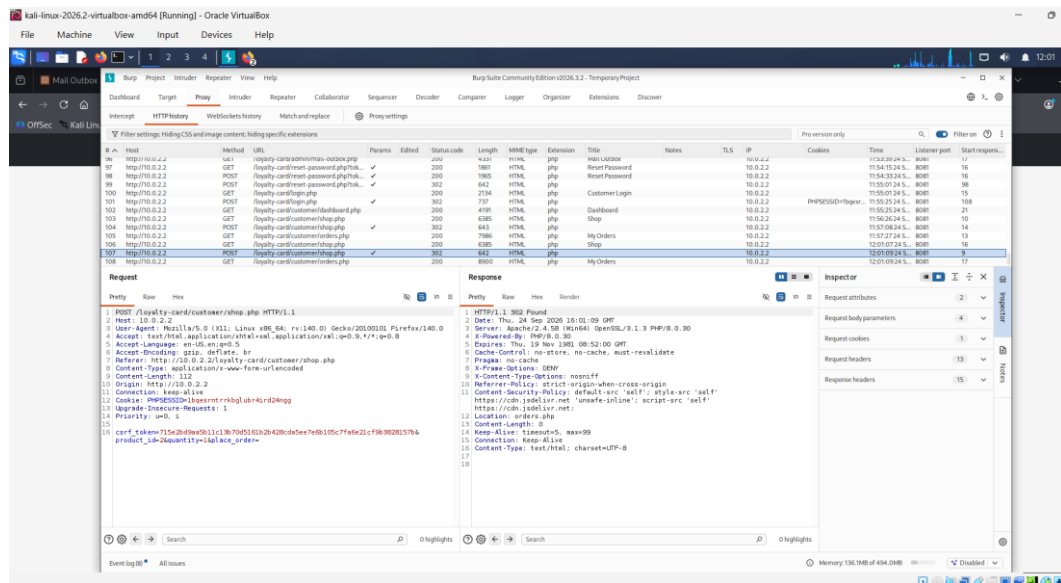


Figure 8.1: Intercepted order-placement request (shop.php) showing only product_id, quantity and csrf_token parameters - no price field is present in the request, confirming that price cannot be tampered with by the client.

8.8 Chapter Summary

This chapter has demonstrated that technical security controls alone - authentication, encryption, and even the CSRF and configuration hardening applied in Chapter 7 - are not sufficient to secure a loyalty-reward platform without a corresponding analysis of its business logic. Price tampering, order-cancellation IDOR, and points double-spending are three vulnerability classes that a conventional, OWASP-Top-Ten-oriented scan would not surface, because each depends on domain-specific knowledge of how the loyalty-points economy is intended to behave rather than on a generic technical payload pattern. By identifying, demonstrating and mitigating each of these flaws through server-side price authority, ownership-scoped database queries, and atomic check-and-set transactions respectively, this chapter extends the project's contribution beyond a standard VAPT assessment and provides a reusable analytical template for assessing the business logic of comparable loyalty- and rewards-based e-commerce platforms.

CHAPTER 9: CONCLUSIONS AND RECOMMENDATIONS

9.1 Conclusion

This project has evaluated and enhanced the security of a locally hosted Loyalty Card Management System through a complete Vulnerability Assessment and Penetration Testing lifecycle that extends beyond assessment into verified remediation and domain-specific business-logic analysis. The baseline assessment, conducted against the OWASP Top Ten using OWASP ZAP, Nikto and SQLMap supported by manual verification in Burp Suite, confirmed that the system correctly resisted SQL injection and XSS but contained critical Cross-Site Request Forgery weaknesses in profile modification, password modification and, most severely, administrative account deletion, together with missing security headers and unhardened session cookies. Every confirmed finding was subsequently remediated at source-code level - anti-CSRF tokens, hardened session cookies, a complete security-header policy, extended brute-force lockout, verified password-reset delivery, mandatory email verification and lightweight bot protection - with the exact changes documented in Chapter 7 to support empirical re-verification.

Beyond this conventional VAPT lifecycle, the project's principal contribution is the business-logic assessment presented in Chapter 8, which identified and mitigated three vulnerability classes specific to the loyalty-points platform - price tampering, order-cancellation IDOR, and points double-spending via race condition - none of which would have been surfaced by a generic OWASP-Top-Ten-oriented assessment. Each was mitigated through a distinct architectural principle: server-side authority over price-determining values, ownership-scoped database queries, and atomic check-and-set transactions respectively. Taken together, the conventional remediation programme and the business-logic contribution demonstrate that combining automated scanning, manual verification, source-level remediation and domain-aware business-logic analysis provides a substantially more complete security assurance process than technical vulnerability scanning alone.

9.2 Recommendations

Complete the Re-Testing Evidence Loop

The remediation documented in Chapter 7 and the mitigations documented in Chapter 8 should be empirically re-verified by repeating the exact ZAP and Burp Suite procedures used at baseline against the remediated application, with the resulting evidence inserted at the points indicated in Sections 7.8, 7.9 and 8.7, to provide direct, reproducible before/after confirmation rather than relying on code review alone.

Enable Full Session Cookie Security Prior to Deployment

The Secure cookie attribute, deliberately left disabled in the local HTTP-only testing environment, should be enabled once the application is deployed over HTTPS, and the Content-Security-Policy should be reviewed against any additional third-party resources introduced after this project.

Extend Business-Logic Testing to Further Workflows

The three business-logic vulnerability classes examined in Chapter 8 were selected as representative, high-impact examples; further workflows - such as bulk order operations or promotional-code stacking,

were introduced - would benefit from the same check-identify-demonstrate-mitigate methodology applied here.

Establish Continuous Security Testing

Security testing should not be treated as a one-time activity; both the technical and business-logic assessments performed in this project should be repeated whenever significant new functionality is introduced, integrating security verification into the ongoing development lifecycle rather than treating it as a discrete, pre-submission exercise.

Adopt a Consistent Shared-Module Pattern for Future Development

The inconsistent application of CSRF protection identified at baseline - present on most forms, absent from one privileged administrative function - illustrates a general risk in incremental application development: a control implemented correctly in most places can still leave the application exposed through a single omission. Future development on this system, and on comparable small-team applications, should treat the shared-module pattern adopted in Chapter 7 as the default approach for any cross-cutting security control, so that adding a new page automatically inherits existing protections rather than requiring each protection to be manually re-applied.

CHAPTER 10: REFLECTION

Undertaking this Vulnerability Assessment and Penetration Testing project has been a profound learning experience, allowing me to bridge theoretical cybersecurity concepts with practical application. Looking back at the process, I recognise how much I have grown both personally and professionally. Initially, I underestimated the time required for manual vulnerability verification and for translating each confirmed finding into a correctly implemented, source-code-level fix; I adapted by creating a structured weekly schedule that separated assessment, remediation and re-testing into distinct phases, which considerably improved my overall project management. I also discovered that my learning is strongest in hands-on, experimental environments, particularly when moving iteratively between identifying a vulnerability, implementing its fix, and immediately verifying the outcome, rather than through reading alone.

The most significant shift in my thinking over the course of this project was recognising that a vulnerability assessment which stops at a list of recommendations is only half of the work; actually implementing and being able to demonstrate that each fix closes the specific exploit path previously demonstrated proved considerably more demanding, and more valuable, than the initial assessment itself. Extending the project into business-logic testing further changed how I think about application security more broadly - I came to see that some of the most damaging vulnerabilities in a system like this are not technical payloads at all, but unchecked assumptions in the workflow logic that only become visible once you deliberately think about the application from an attacker's perspective rather than a developer's. If I were to undertake this project again, I would allocate more time to the initial environment and remediation-evidence tooling so that before/after verification could be captured continuously throughout the project rather than concentrated at the end. This reflective process has equipped me with the analytical mindset and project-management discipline needed to approach future cybersecurity work with greater confidence.

REFERENCES

- [1] Ö. Aslan, S. Aktuğ, M. E. R. V. E. Özkan Okay, A. Yilmaz, and E. Akin, "A comprehensive review of cyber security vulnerabilities, threats, attacks, and solutions," *Electronics*, vol. 12, no. 6, 2023.
- [2] M. Otieno, D. Odera, and J. E. Ounza, "Theory and practice in secure software development lifecycle: A comprehensive survey," *World J. Adv. Res. Rev.*, vol. 18, no. 3, pp. 53-78, 2023.
- [3] A. Alquwayzani, R. Aldossri, and M. Frikha, "Mitigating security risks in firewalls and web applications using vulnerability assessment and penetration testing (VAPT)," *Int. J. Adv. Comput. Sci. Appl.*, vol. 15, no. 5, p. 1348, 2024.
- [4] U. Ravindran and R. V. Potukuchi, "A review on web application vulnerability assessment and penetration testing," *Rev. Comput. Eng. Stud.*, vol. 9, no. 1, p. 1, 2022.
- [5] M. Nawrocki and J. Kołodziej, "Vulnerabilities of web applications: Good practices and new trends," *Appl. Cybersecurity Internet Governance*, vol. 3, no. 2, pp. 122-143, 2024.
- [6] S. S. Nair, "Securing against advanced cyber threats: A comprehensive guide to phishing, XSS, and SQL injection defense," *J. Comput. Sci. Technol. Stud.*, vol. 6, no. 1, pp. 76-93, 2024.
- [7] B. Riskhan, M. A. U. Sheikh, M. S. Hossain, K. Hussain, Z. Zainol, and N. Z. Jhanjhi, "Major vulnerabilities of web application in real world scenarios and their prevention," in *Proc. 2025 Int. Conf. Intell. Cloud Comput. (ICoICC)*, May 2025, pp. 1-6.
- [8] N. Odeh and S. Hijazi, "Detecting and preventing common web application vulnerabilities: A comprehensive approach," *Int. J. Inf. Technol. Comput. Sci.*, vol. 15, no. 3, p. 2641, 2023.
- [9] D. Rohmaniah, W. M. Ashari, L. Lukman, and A. D. Putra, "Enhancing website security using vulnerability assessment and penetration testing (VAPT) based on OWASP Top Ten," *J. Appl. Informatics Comput.*, vol. 9, no. 2, pp. 404-411, 2025.
- [10] D. F. Priambodo, A. D. Rifansyah, and M. Hasbi, "Penetration testing Web XYZ berdasarkan OWASP risk rating," *Teknika*, vol. 12, no. 1, pp. 33-46, 2023.
- [11] A. Joshi, A. Dumka, D. P. Singh, and A. Semwal, "Comprehensive survey on vulnerability assessment and penetration testing (VAPT) for cloud-hosted web applications," in *Proc. Int. Conf. Sustainable Business Practices Innovative Models (ICSBPIM)*, Nov. 2025, p. 278.
- [12] K. S. R. Rodriguez, J. D. A. Galindo, J. T. Juan, J. R. B. Higuera, J. B. Higuera, and J. A. S. Montalvo, "Secure development methodology for full stack web applications: Proof of the methodology applied to Vue.js, Spring Boot and MySQL," *Comput. Mater. Continua*, vol. 85, no. 1, pp. 1807-1858, 2025.
- [13] J. R. Tadhani, V. Vekariya, V. Sorathiya, S. Alshathri, and W. El-Shafai, "Securing web applications against XSS and SQLi attacks using a novel deep learning approach," *Sci. Rep.*, vol. 14, no. 1, p. 1803, 2024.
- [14] P. S. S. K. Gandikota, D. Valluri, S. B. Mundru, G. K. Yanala, and S. Sushaini, "Web application security through comprehensive vulnerability assessment," *Procedia Comput. Sci.*, vol. 230, pp. 168-182, 2023.
- [15] S. F. Wen and B. Katt, "A quantitative security evaluation and analysis model for web applications based on OWASP application security verification standard," *Comput. Secur.*, vol. 135, p. 103532, 2023.
- [16] P. Lachkov, L. A. Tawalbeh, and S. Bhatt, "Vulnerability assessment for applications security through penetration simulation and testing," *J. Web Eng.*, vol. 21, no. 7, pp. 2187-2208, 2022.
- [17] L. Dencheva, "Comparative analysis of static application security testing (SAST) and dynamic application security testing (DAST) by using open-source web application penetration testing tools," M.S. thesis, National College of Ireland, Dublin, 2022.

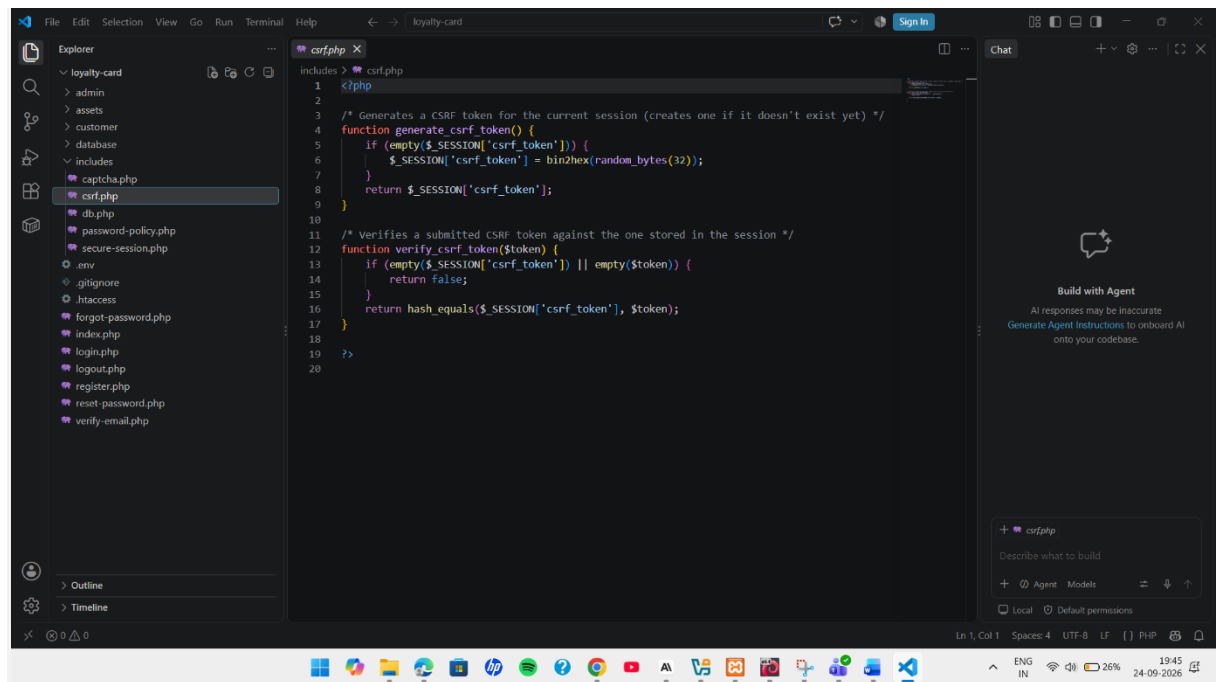
- [18] R. P. K. Kollepalli, M. J. S. Reddy, B. L. Sai, A. Natarajan, S. Mathi, and V. Ramalingam, "An experimental study on detecting and mitigating vulnerabilities in web applications," *Int. J. Safety Secur. Eng.*, vol. 14, no. 2, 2024.
- [19] Z. Li, Q. Zhang, and G. Yang, "EPPTA: Efficient partially observable reinforcement learning agent for penetration testing applications," *Eng. Rep.*, vol. 7, no. 1, p. e12818, 2025.
- [20] A. S. George and S. Sagayarajan, "Securing cloud application infrastructure: Understanding the penetration testing challenges of IaaS, PaaS, and SaaS environments," *Partners Universal Int. Res. J.*, vol. 2, no. 1, pp. 24-34, 2023.
- [21] R. Bakır, "UniEmbed: A novel approach to detect XSS and SQL injection attacks leveraging multiple feature fusion with machine learning techniques," *Arabian J. Sci. Eng.*, vol. 50, no. 19, pp. 15591-15604, 2025.
- [22] O. I. Omotosho, Y. P. Lawal, B. I. Ogunrinola, O. L. Ajayi, T. A. Adediji, and A. A. Akintunde, "Enhancing network security using vulnerability assessment and penetration testing," *FUOYE J. Eng. Technol.*, vol. 10, no. 1, pp. 76-83, 2025.
- [23] W. Zhang, J. Xing, and X. Li, "Penetration testing for system security: Methods and practical approaches," *arXiv:2505.19174*, 2025.
- [24] S. Elder, N. Zahan, R. Shu, M. Metro, V. Kozarev, T. Menzies, and L. Williams, "Do I really need all this work to find vulnerabilities? An empirical case study comparing vulnerability detection techniques on a Java application," *Empir. Softw. Eng.*, vol. 27, no. 6, p. 154, 2022.
- [25] A. Akinsola and A. Akinde, "Enhancing software supply chain resilience: Strategy for mitigating software supply chain security risks and ensuring security continuity in development lifecycle," *arXiv:2407.13785*, 2024.

APPENDIX A: KEY SOURCE-CODE LISTINGS

This appendix reproduces the final source code of the four shared security modules referenced throughout Chapter 7, each of which is included by every relevant page in the application to ensure the corresponding control is applied consistently.

A.1 CSRF Token Handling (includes/csrf.php)

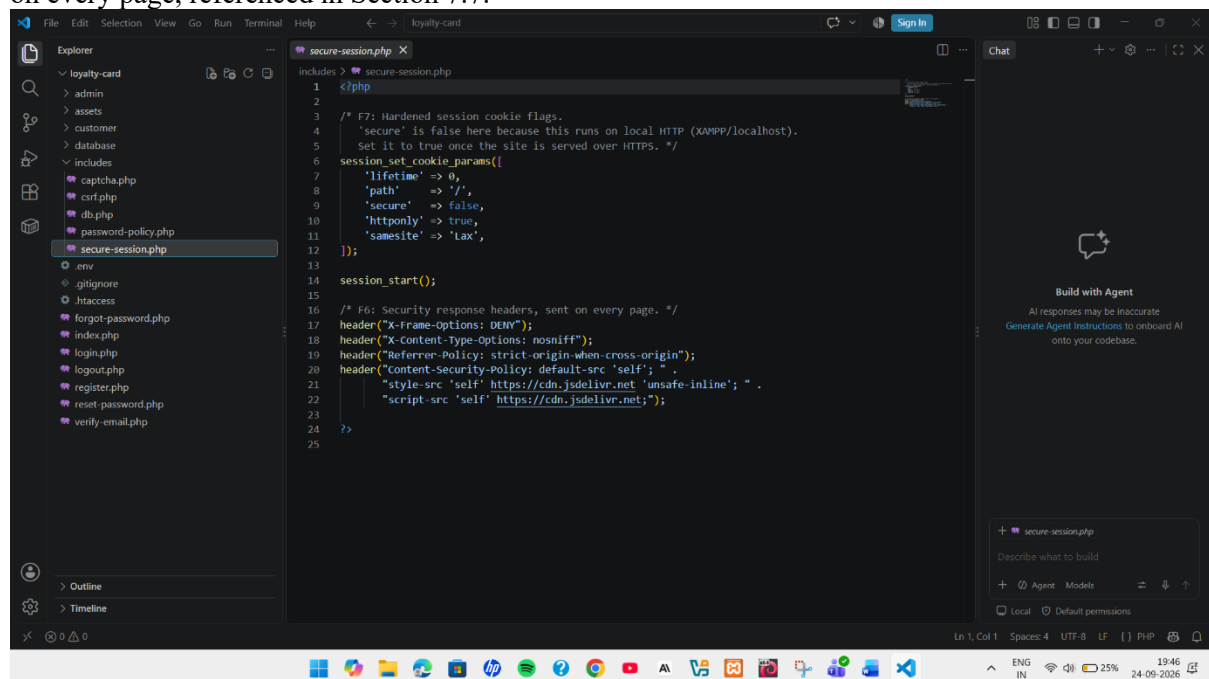
Generates a session-bound, cryptographically random CSRF token and verifies submitted tokens using a constant-time comparison, referenced in Section 7.3.



```
1 <?php
2
3 /* Generates a CSRF token for the current session (creates one if it doesn't exist yet) */
4 function generate_csrf_token() {
5     if (empty($_SESSION['csrf_token'])) {
6         $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
7     }
8     return $_SESSION['csrf_token'];
9 }
10
11 /* Verifies a submitted CSRF token against the one stored in the session */
12 function verify_csrf_token($token) {
13     if (empty($_SESSION['csrf_token']) || empty($token)) {
14         return false;
15     }
16     return hash_equals($_SESSION['csrf_token'], $token);
17 }
18
19 ?>
```

A.2 Secure Session Bootstrap (includes/secure-session.php)

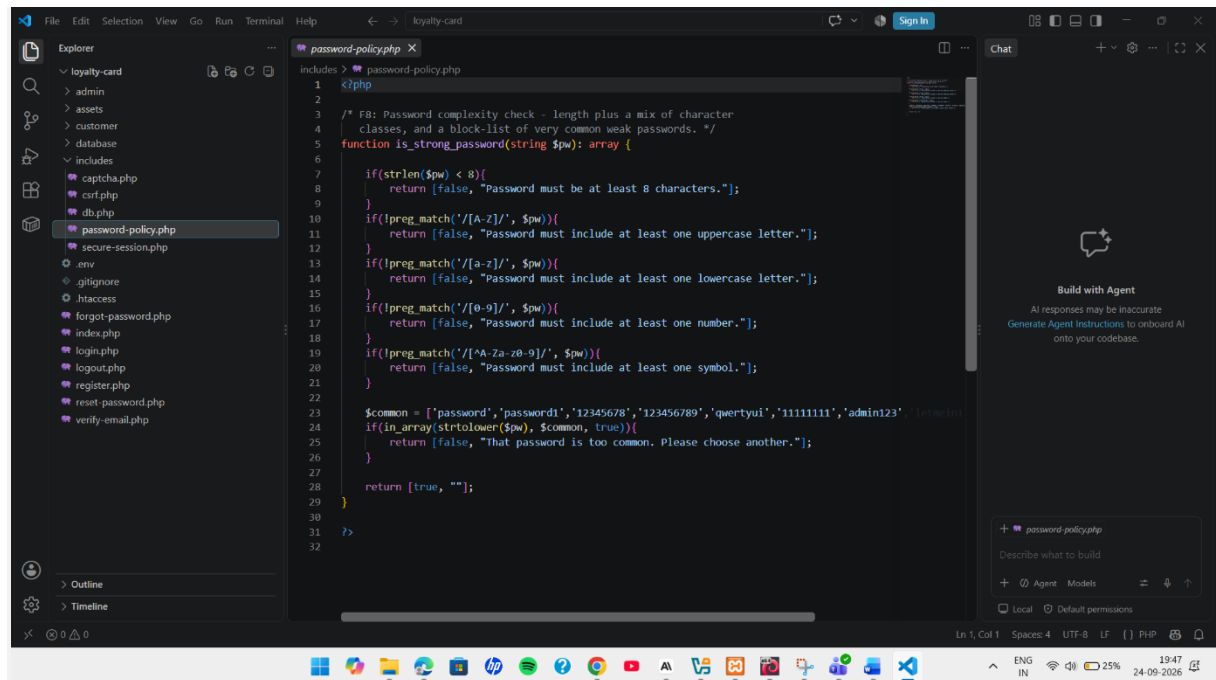
Hardens the session cookie (HttpOnly, SameSite) and sends the application's security response headers on every page, referenced in Section 7.7.



```
1 <?php
2
3 /* F7: Hardened session cookie flags.
4  'secure' is false here because this runs on local HTTP (XAMPP/localhost).
5  Set it to true once the site is served over HTTPS. */
6 session_set_cookie_params([
7     'lifetime' => 0,
8     'path' => '/',
9     'secure' => false,
10    'httponly' => true,
11    'samesite' => 'lax',
12]);
13
14 session_start();
15
16 /* F6: Security response headers, sent on every page. */
17 header("X-Frame-Options: DENY");
18 header("X-Content-Type-Options: nosniff");
19 header("Referrer-Policy: strict-origin-when-cross-origin");
20 header("Content-Security-Policy: default-src 'self'; " .
21     "style-src 'self' https://cdn.jsdelivr.net 'unsafe-inline'; " .
22     "script-src 'self' https://cdn.jsdelivr.net;");
23
24 ?>
```

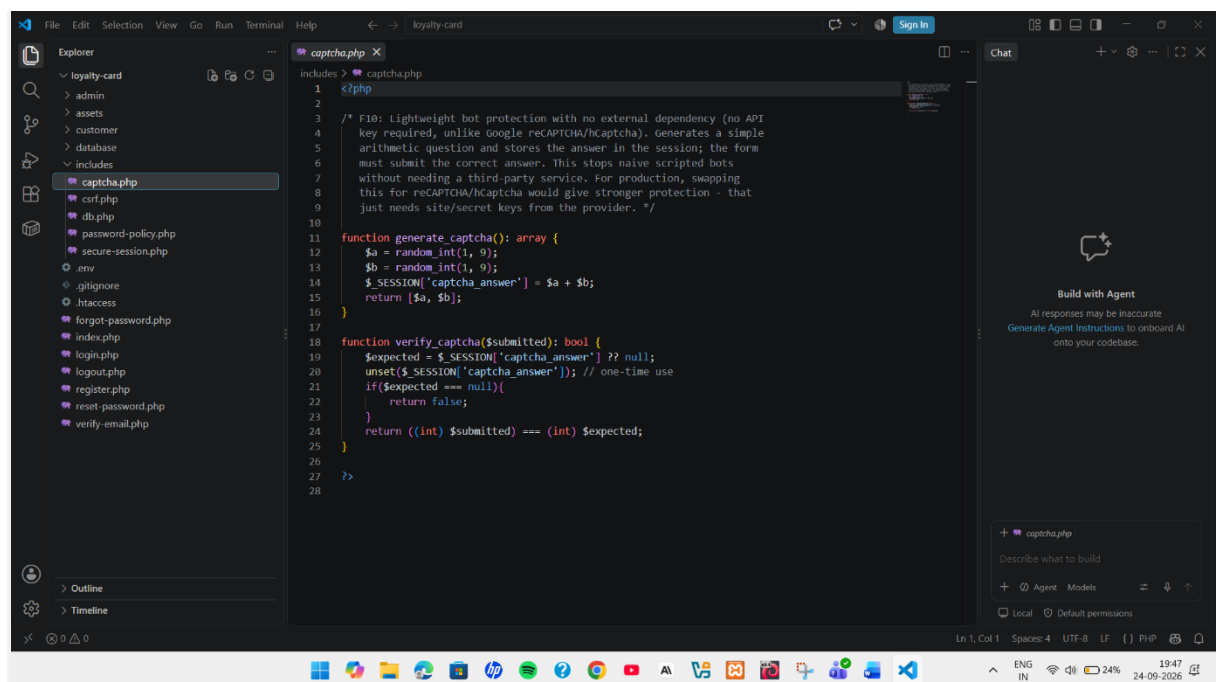
A.3 Password Complexity Policy (includes/password-policy.php)

Enforces a minimum length and character-class complexity requirement, and blocks a short list of common weak passwords, referenced in Section 7.6.



A.4 Lightweight CAPTCHA (includes/captcha.php)

Generates and verifies a session-bound arithmetic challenge used for bot protection on public forms, referenced in Section 7.5.4.



APPENDIX B: SUPPLEMENTARY TESTING EVIDENCE

This appendix presents additional evidence captured during re-testing (Chapter 7) that supports, but was not essential to include in, the main chapter narrative.

B.1 Full Post-Remediation ZAP Alert Summary

The complete list of alerts raised by the post-remediation OWASP ZAP automated scan (Section 7.8), shown here for completeness alongside the individually discussed High-risk SQL Injection false positive (Section 7.3).

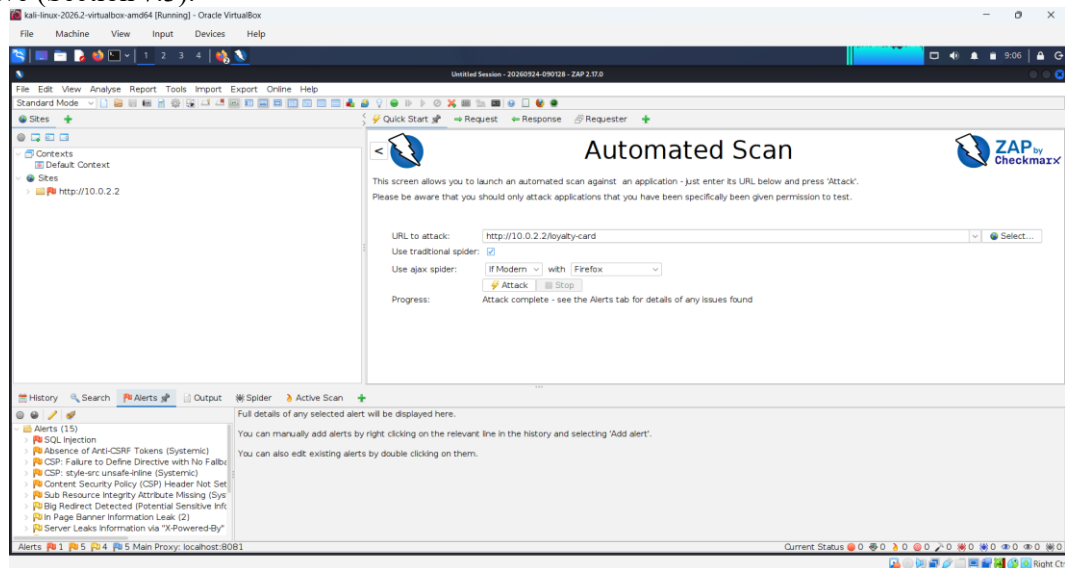


Figure B.1: Post-remediation ZAP scan summary - Attack complete, Alerts (15): 1 High, 5 Medium, 4 Low, 5 Informational.

B.2 Rate-Limiting Table Evidence

Database-level evidence supporting the brute-force lockout test (Section 7.5.1): five failed customer-login attempts recorded in the `rate_limits` table within a one-minute window, immediately before the lockout message shown in Figure 7.6 was triggered.

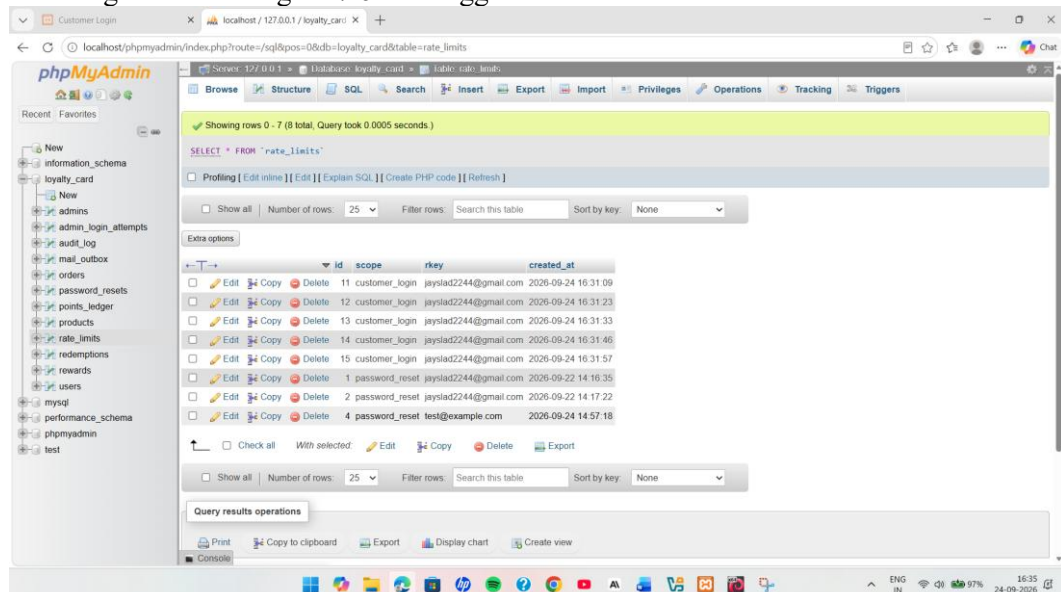


Figure B.2: `rate_limits` table in phpMyAdmin showing five recorded customer_login attempts for the test account within the fifteen-minute lockout window.