THREAT MODEL

This is a threat model for the "Logie's" service application

Adam Logan

27/04/2023

Queen's University Belfast

Table of Contents

1. EXECUTIVE SUMMARY

This report presents an analysis of potential threats to the "Logie's" bar and restaurant service application, which allows customers to place online orders for delivery to their home or table. The application is built using the LAMP (Linux, Apache, MySQL, PHP) architecture. To identify and mitigate potential threats, various documentation such as the architecture diagram, ER (Entity Relationship) diagram, and a DFD (Data Flow Diagram) were reviewed for security concerns.

The primary concerns of the business are protecting customer information and safeguarding their reputation. As such, the threat identification process focused on these areas. Six threats were identified, and the report details how attackers can exploit these to cause damage to the business, as well as how the business can counteract these threats.

All diagrams within the report can be found here (a diagram.net account is needed to view all diagrams) and all tables can be found here.

2. APPROACH TAKEN

The report outlines a methodological approach to threat modelling with the aim of enhancing vulnerability detection and accurately assessing the threat level. The process commences with the identification of assets requiring protection and the determination of entry points to the system. The STRIDE methodology from Microsoft is employed for threat identification, offering greater flexibility than the 'threat list' methodology in detecting previously unidentified threats and facilitating their categorization. The risk of a particular threat is assessed using the DREAD methodology, with each category rated between 0 and 3 and higher scores indicating greater risk to the business. The scores are then summed to classify the threat as low, medium, or high risk.

3. BUSINESS ARCHITECTURE

The below business and application were created with the assistance of Dean Logan and Chris Magee. The architecture diagram, dataflow and ER diagram may differ between students.

3.1 The Business and Application

The application is designed to assist a local bar and restaurant in improving service speed and offering a delivery service through the use of a website. Customers ordering within the restaurant can access the site by scanning a QR code located on their table. Upon placement of an order, the details are displayed in the kitchen and subsequently delivered to the customer.

3.2 Technology Used

LAMP is the chosen technology stack for this application, as can be seen in the Architecture diagram in figure 3.2.1.

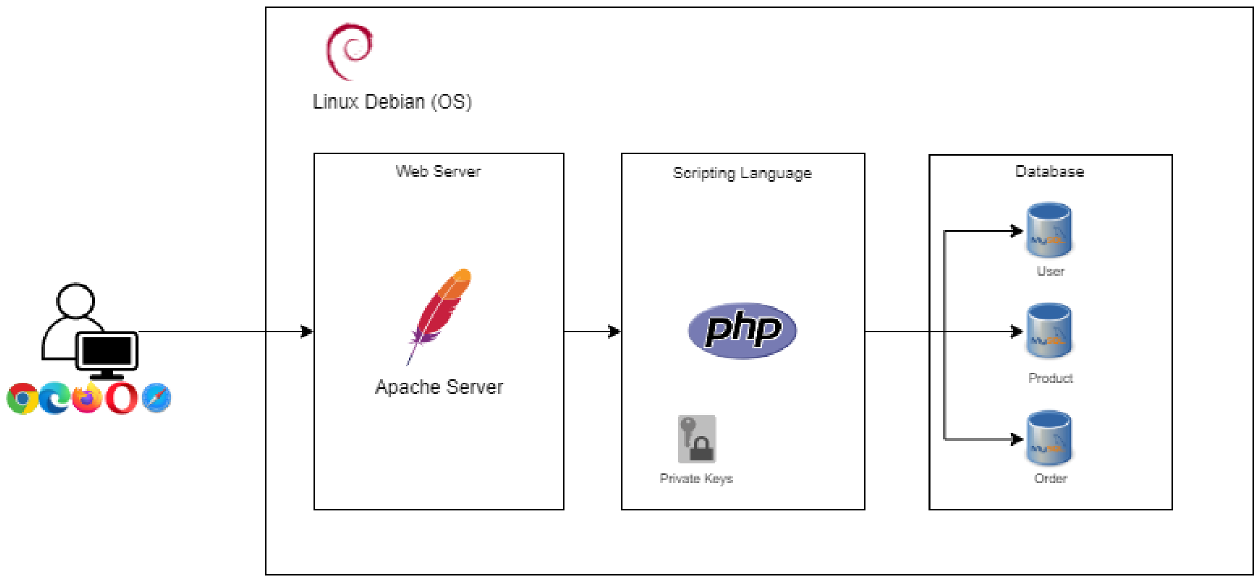

Figure 3.2.1: Architecture diagram of the system showing the technologies used

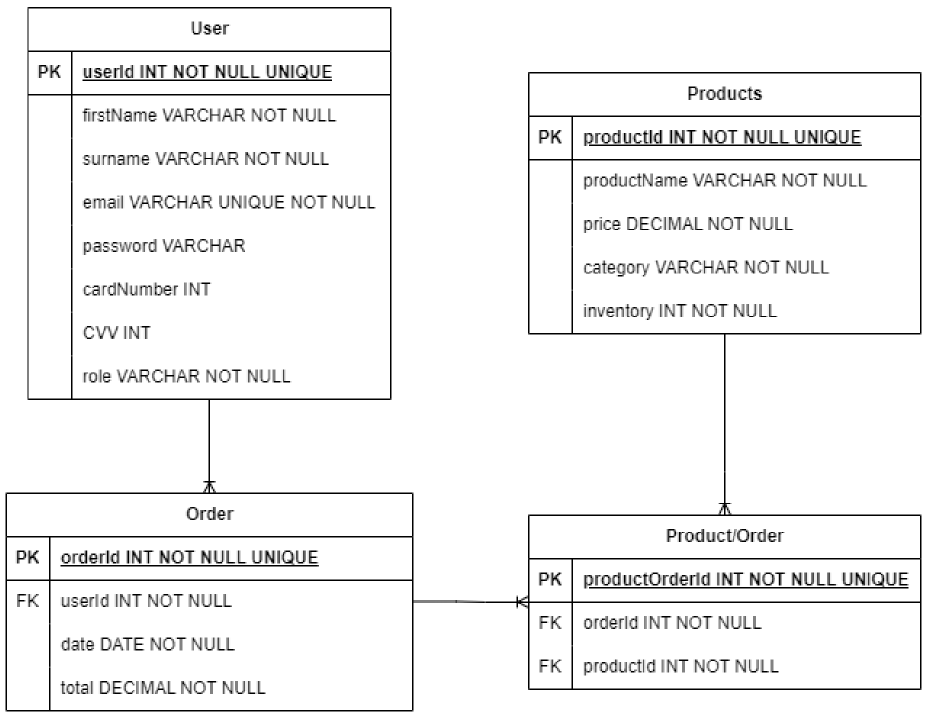
User
PK userId INT NOT NULL UNIQUE
firstName VARCHAR NOT NULL
surname VARCHAR NOT NULL
email VARCHAR UNIQUE NOT NULL
password VARCHAR
cardNumber INT
CVV INT
role VARCHAR NOT NULL
Products
PK productId INT NOT NULL UNIQUE
productName VARCHAR NOT NULL
price DECIMAL NOT NULL
category VARCHAR NOT NULL
inventory INT NOT NULL
Order
PK orderId INT NOT NULL UNIQUE
FK userId INT NOT NULL
date DATE NOT NULL
total DECIMAL NOT NULL
Product/Order
PK productOrderId INT NOT NULL UNIQUE
FK orderId INT NOT NULL
FK productId INT NOT NULL

Figure 3.2.2: ER diagram of the database, for the system

3.3 Data Flow

Users must provide a username, password, and payment information, which is stored in a database and protected using the SHA-256 hashing algorithm. User orders are stored in the database and displayed to kitchen staff.

A DFD for the ordering process can be seen in figure 3.3.1.

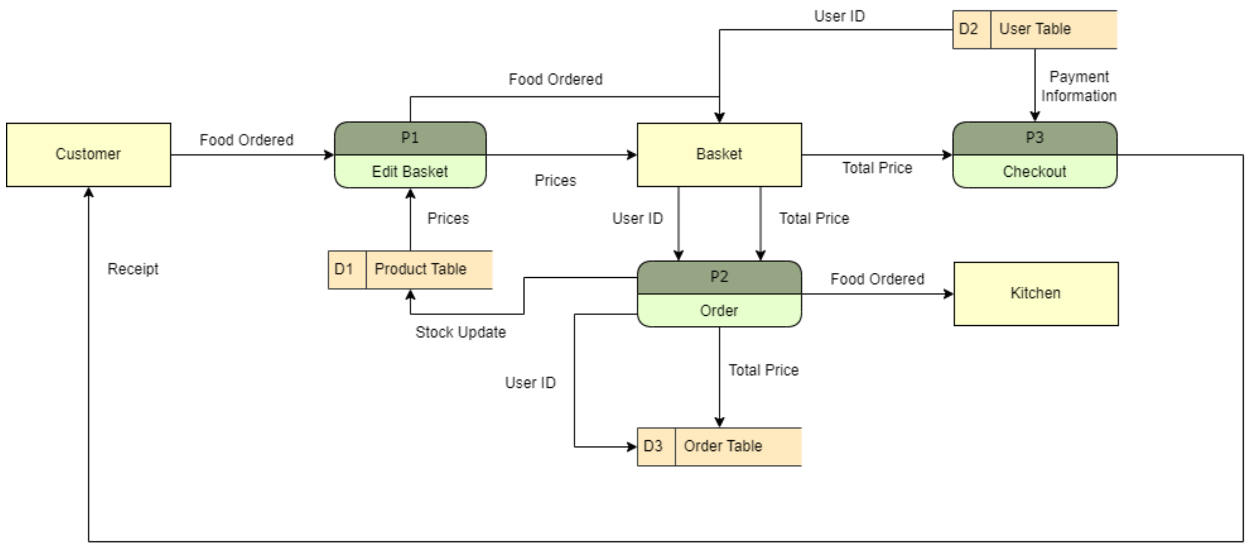

Figure 3.3.1: A data flow diagram of the process of a customer ordering food

4. ARCHITECTURE DECOMPOSITION

4.1 Identified Assets and Entry Points

The protection of customer information is a crucial asset for the application. Ensuring the confidentiality and integrity of payment information (A1) is essential to prevent malicious actors from altering it and fraudulently charging innocent customers. Additionally, safeguarding customer addresses (A2) is necessary to prevent potential harassment or use in future social engineering attacks. Another key asset for the business is the site (A3) itself, which must be protected from defacement or denial of access by malicious actors seeking to damage the brand.

The system's probable entry points include user interactions through data entry forms and HTTP requests.

4.2 Authentication/Authorisation

The authentication mechanism of the application involves a rudimentary username-password system and additionally integrates Single Sign-On (SSO) through Google, Facebook, or Microsoft accounts to enhance user experience and security. Despite the possibility of associated risks, the implementation of this approach can offer advantages due to the greater security budgets of these companies.

4.3 Decomposition Diagram

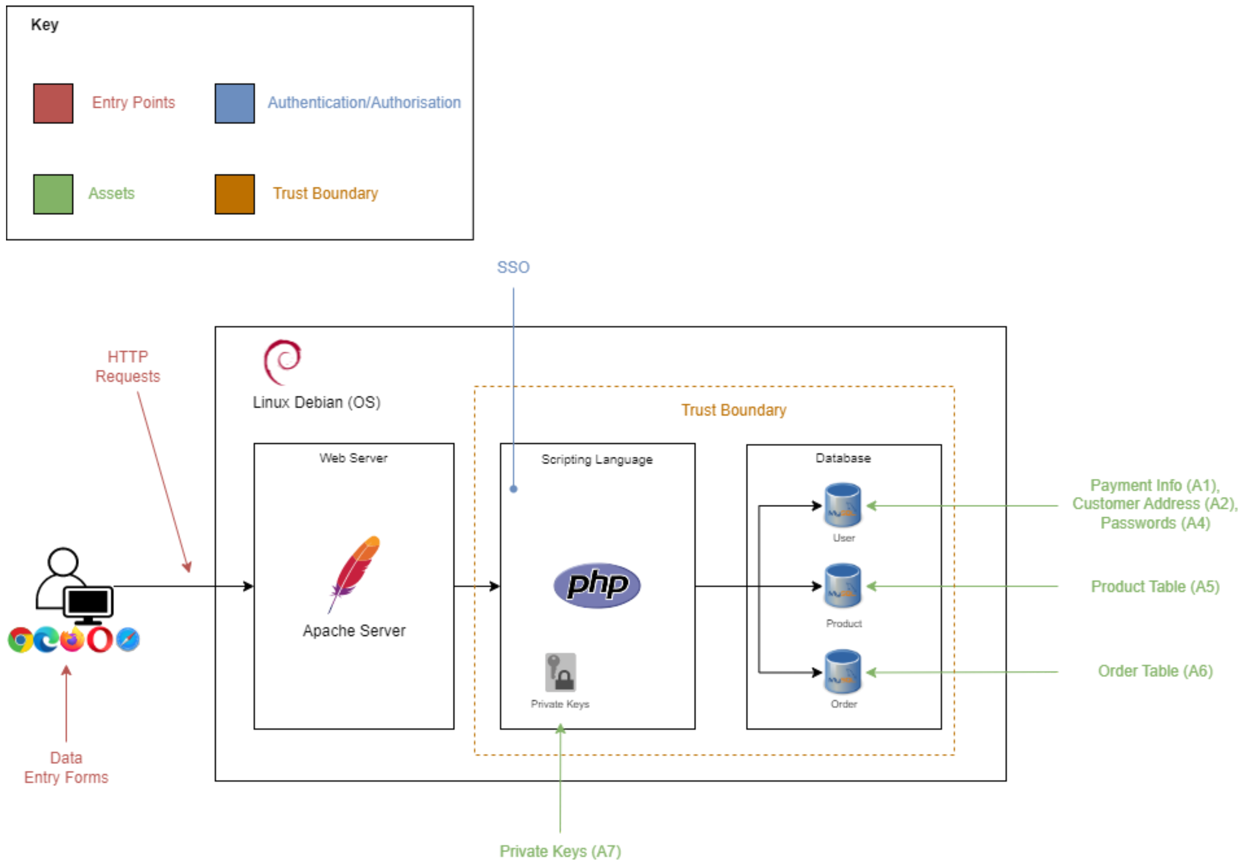

Figure 3.3.1: Architecture decomposition diagram displaying the entry points, trust boundary, assets and authentication/authorisation of the system

5. THREAT IDENTIFICATION

THREAT ID	TA1
THREAT TITLE	SQL Injection (SQLi)
THREAT AGENT	External - Lone Individual
THREAT DESCRIPTION	As the name implies, this is when SQL code is injected into the system. From this the attacker is able to view and/or delete data from the database.
THREAT TARGET	Customer information such as passwords, payment information and address.
ATTACK SURFACE	Through data entry forms, such as the login page or directly through requests to the website.
ATTACK TECHNIQUES	There are several ways a SQLi can be performed but the simplest way is to add an apostrophe before the SQL code you would like to input and a SQL comment '--' after the code.
LIKELIHOOD	High: Although this has decreased in popularity it is still in third position in the OWASP top 10 [18]. As there is a wide availability of automated tools which perform SQLi [25][26] increases the likelihood.
IMPACT	High: There would be damage to the reputation of the business if information is leaked or service may be interrupted if the payment data or addresses is deleted.
MITIGATION	- Change the default configuration of MySQL [1] - Ensure that hashed passwords are salted - Employ the principle of least privilege [17] - Move towards a framework which employs ORM (Object Relational Mapping), such as Django
CONTROL	- Use stored procedures [17] - Use parameterised statements [12][28] - Regular expressions can be used to filter input [23]

THREAT ID	TA2
THREAT TITLE	Cross-Site Scripting (XSS)
THREAT AGENT	External - Lone Individual
THREAT DESCRIPTION	Cross-Site Scripting (XSS) is an injection attack that involves the injection of JavaScript code. The attacker's injected code is executed within the browser, providing a significant amount of access to the application. There are three main types of XSS: Persistent/Stored XSS, Non-Persistent/Reflected XSS, and DOM-Based XSS [7].
THREAT TARGET	Customer information, a user's session and the site itself.
ATTACK SURFACE	Through data entry forms, and HTTP requests when a Reflected XSS is performed.
ATTACK TECHNIQUES	Techniques include inserting script tags into data entry forms [12] and manipulating HTTP requests.
LIKELIHOOD	High: Simple to carry out, wide availability of scanner tools such as Burp Suite and XSStrike [20] and the existence of readily available scripts.
IMPACT	High: Ranges from site defacement to session hijacking [21].
MITIGATION	- Implementing a content security policy within the HTTP responses to disallow inline JavaScript [12][15]
CONTROL	- Use entity encoding [12] for special characters - To use PHP character escaping functions such as htmlspecialchars() [24] - To switch to the React framework as this automatically escapes user input

THREAT ID	TA3
THREAT TITLE	Distributed Denial Of Service (DDos)
THREAT AGENT	External - Criminal Organisation
THREAT DESCRIPTION	A DDoS attack is when a server is overwhelmed by flooding it with traffic from multiple sources, making it unavailable to legitimate users.
THREAT TARGET	To deny service to the site.
ATTACK SURFACE	The web server hosting the application.
ATTACK TECHNIQUES	A botnet is most commonly used to deploy a DDos attack and the following some of the methods used to overwhelm the server are TCP SYN flood, ICMP flood and UDP flood [29].
LIKELIHOOD	Medium: As a large number of resources are needed to carry this out and access to a botnet is not specifically required although this increases the chances of success. However, the likelihood is increased due to the availability of DDos attack tools [3]
IMPACT	High: Lost revenue, as the site will be down, and damage to the reputation of the business.
MITIGATION	- Graceful degradation [16] to reduce the effect upon ongoing processes - Incorporating IPS (Intrusion Prevention System) and IDS (Intrusion Detection System) [6]
CONTROL	- Incorporating a load balancer to manage traffic - By using a cloud provider, such as AWS and Azure, their in-built DDos protections will assist in mitigation

THREAT ID	TA4
THREAT TITLE	HTTP Parameter Pollution (HPP)
THREAT AGENT	External - Lone Individual
THREAT DESCRIPTION	Duplicated parameters in a HTTP request can result in unexpected behaviour of an application and may be exploited by attackers [10].
THREAT TARGET	Business processes.
ATTACK SURFACE	Endpoints of the system.
ATTACK TECHNIQUES	PHP and Apache use the last occurrence of a given parameter [4]. This means that additional parameters can be added to the end of the input of a parameter by using the '&' character to denote the end of an argument and the start of another parameter. For example, a user can add new arguments by inputting 'param1=arg1&param2=arg2&param1=maliciousarg'.
LIKELIHOOD	Low: As error messages from endpoints are less verbose [28] the vulnerability is harder to detect, and extensive knowledge of the system is required to pass the correct arguments. This is also compounded by the fact that this is not a widely known attack [2]
IMPACT	Medium: Allows malicious actors to discount their meal by changing the 'total' parameter to zero and therefore damage the business through lost revenue.
MITIGATION	- Encrypting the arguments, prevents a user from entering plain text and therefore is required to first discover the encryption technique used to send valid data
CONTROL	- By blacklisting the '&' and '?' character for user input a malicious actor is unable to add more parameters - The 'entity encoding' technique discussed in the 'control' section for threat TA2 can also apply here

THREAT ID	TA5
THREAT TITLE	HTML Injection
THREAT AGENT	External - Lone Individual
THREAT DESCRIPTION	A HTML injection attack involves an attacker injecting malicious HTML code into a web page [9], with the intention of manipulating its content to trick users into revealing sensitive information or taking malicious actions.
THREAT TARGET	Customer information.
ATTACK SURFACE	Through data entry forms, and HTTP requests.
ATTACK TECHNIQUES	Typically, a similar technique to XSS is used for HTML injection although the difference lies within how the injected code is typically used. A distinguishing factor of this attack over XSS is how the end user is targeted. A malicious actor will most likely perform content spoofing of the login page.
LIKELIHOOD	Low: Although not difficult to perform, it is more difficult to deceive users and therefore other attacks may be chosen instead
IMPACT	Medium: Tricked users will reveal their information to an attacker
MITIGATION	- By educating both customers and staff about the possibility of content spoofing this will decrease those ‘tricked’ by a malicious actor
CONTROL	- As with the threats (TA1, TA2 and TA4) input sanitisation techniques, such as blacklisting, entity encoding and escaping characters are the best defence against this attack

THREAT ID	TA6
THREAT TITLE	Cross-Site Request Forgery (XSRF)
THREAT AGENT	External - Criminal Organisation/Lone Individual
THREAT DESCRIPTION	XSRF is a social engineering technique that utilises the authenticated user's credentials to perform unauthorised actions on a web application. This attack can be initiated by visiting a malicious website or through email-based links.
THREAT TARGET	Possible user actions.
ATTACK SURFACE	Endpoints of the system.
ATTACK TECHNIQUES	The attack is typically executed through social engineering methods, which deceive users into clicking on a malicious link, often in the form of a GET request. These links can be delivered through emails [19] or embedded images [22].
LIKELIHOOD	Medium: Although technically simple, this requires social engineering tactics and knowledge of the system.
IMPACT	High: Any action a valid user performs through HTTP requests, the attacker can as well
MITIGATION	- Requiring user interaction, such as through a CAPTCHA [27] - Utilise POST requests instead of GET requests [12] - By using hidden fields within a html form, it is possible to force a user to use your form [22]
CONTROL	- Including the session ID directly in the request, therefore an attacker requires the session ID of the user to form the malicious request [8] - Using the shared token technique by implementing a hidden token field, which is generated by the session ID [8] - A proxy between the web server and the target application, which holds a token table which maps tokens to session IDs can be used to implement the above [8]

6. THREAT RATING / RISK ASSESSMENT

The following tables can also be found here.

THREAT ID	SPOOFING IDENTITY	TAMPERING	REPUDIATION	INFORMATION DISCLOSURE	DENIAL OF SERVICE	ELEVATION OF PRIVILAGE
TA1	☑	☑	☑	☑	☑	☑
TA2	☑	☐	☑	☐	☐	☑
TA3	☐	☐	☐	☐	☑	☐
TA4	☐	☑	☐	☑	☐	☐
TA5	☐	☑	☐	☑	☐	☐
TA6	☑	☑	☐	☑	☐	☐

THREAT ID	Damage	Reproducibility	Exploitability	Affected users	Discoverability	Total	Rating
TA1	3	3	2	3	3	14	High
TA2	3	2	2	3	3	13	High
TA3	3	1	1	3	2	10	Medium
TA4	2	3	1	1	0	7	Medium
TA5	2	2	2	1	3	10	Medium
TA6	2	3	2	1	1	9	Medium

Key: Low (0 - 5), Medium (6 - 10) and High (11 - 15)

7. TRACEABILITY / COMPLIANCE MATRIX

THREAT AGENT	ASSET	ATTACK	ATTACK SURFACE	ATTACK GOAL	IMPACT	CONTROL	MITIGATION
TA1	A1, A2, A4	SQLi	Login page	Login as another user, delete data or reveal passwords	High: Passwords revealed or logged in as admin	- Stored Procedures - Parameterised Statements - Filter input	- Change default configurations - Salt hashed passwords - Switch to a ORM framework
TA2	A3, A4	XSS	Data entry forms and HTTP requests	To gain control of a user's session	High: Site defacement to session hijacking	- Escape characters - Switch to React framework	- Implement a content security policy
TA3	A3	DDos	The server	To prevent access to the site	High: Site cannot be accessed	- Load balancer - Migrate to cloud	- Graceful degradation - IPS/IDS
TA4	A6	HPP	HTTP requests	Change arguments of HTTP requests to desired values	Medium: Individuals may discount meals	- Blacklist - Use entity encoding	- Encrypting the arguments
TA5	A4	HTML Injection	Login page	Trick users into revealing passwords	Medium: Reveals customer information	- Character encoding - Escape characters - Blacklist	- Educate users on security
TA6	A1	XSRF	HTTP requests	Trick users into performing unwanted actions	High: Attacker can perform any action through HTTP requests	- Require user interaction - Use hidden fields within the html form	- Include the session ID directly within the request - Shared token technique

8. CONCLUSION

The threat model identifies various threats to the application, with a focus on obtaining customer information such as payment details and passwords. Input sanitation is an effective control technique against most of these threats.

Incorporating security within software artefacts [13] and adopting a DevSecOps model [14] throughout the SDLC (Software Development Lifecycle) can reduce the likelihood of ignoring these threats. The TDD (Test-Driven Development) model can test the effectiveness of threat control measures during development [11].

Additional measures recommended for the application's security include implementing IPS, IDS, WAF (Web Application Firewall) [5], load balancer, and migrating the application to the cloud. Raising awareness among staff and customers about potential social engineering attacks is also deemed essential, and changing default configurations is imperative [1].

9. REFERENCES

[1] Andress, Jason, "Operating System Hardening." *Foundations of Information Security: A Straightforward Introduction*, No Starch Press, 2019, pp. 146 - 150.

[2] Balduzzi, Marco, et al. "Automated Discovery of Parameter Pollution Vulnerabilities in Web Applications." *ResearchGate*, May 2011, https://www.researchgate.net/publication/221655534_Automated_Discovery_of_Parameter_Pollution_Vulnerabilities_in_Web_Applications#fullTextFileContent Accessed 27 April 2023.

[3] Bhosale, Karuna S., et al. *The distributed denial of service attacks (DDoS) prevention mechanisms on application layer*. Edited by Bratislav D. Milovanović, IEEE, 2017, https://ieeexplore.ieee.org/document/8246247 Accessed 20 April 2023.

[4] Carettoni, Luca, and Stefano di Paola. "HTTP Parameter Pollution." *OWASP EU09 Poland*, https://owasp.org/www-pdf-archive/AppsecEU09_CarettoniDiPaola_v0.8.pdf Accessed 15 April 2023.

[5] Clincy, Victor, and Hossain Shahriar. *Web Application Firewall: Network Security Models and Configuration*. Edited by Sorel Reisman, IEEE, 2018, https://ieeexplore.ieee.org/document/8377769 Accessed 25 April 2023.

[6] Corrêa, João Henrique G. M., and Epaminondas A. Sousa Junior. *Selectivity and Autoscaling as Complementary Defenses for DDoS Protection to Cloud Services*. IEEE, 2019, https://ieeexplore.ieee.org/document/9064139 Accessed 20 April 2023.

[7] Gupta, Shashank, and B. B. Gupta. “Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art.” *Springer*, 14 September 2025, https://rdcu.be/daSRT Accessed 12 April 2023.

[8] Jovanovic, Nenad, et al. *Preventing Cross Site Request Forgery Attacks*. IEEE, 2006, https://ieeexplore.ieee.org/document/4198791 Accessed 24 April 2023.

[9] Keary, Eoin. “Testing for HTML Injection.” *WSTG - Latest | OWASP Foundation*, https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection Accessed 19 April 2023.

[10] Keary, Eoin. “Testing for HTTP Parameter Pollution.” *WSTG - Latest | OWASP Foundation*, https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/04-Testing_for_HTTP_Parameter_Pollution Accessed 18 April 2023.

[11] Magda Chelly, et al., “Secure coding and secure software development.” *Building a Cyber Resilient Business: A Cyber Handbook for Executives and Boards*, Packt Publishing, Limited, 2022, pp. 137 - 139.

[12] McDonald, Malcolm. *Web Security for Developers: Real Threats, Practical Defense*. No Starch Press, 2020.

[13] McGraw, Gary. *Software security*. no. 2, IEEE, 2 August 2004, https://ieeexplore.ieee.org/document/1281254 Accessed 22 April 2023.

[14] NIST. “DevSecOps | CSRC.” *NIST Computer Security Resource Center*, 21 October 2020, https://csrc.nist.gov/projects/devsecops Accessed 24 April 2023.

[15] OWASP. “Content Security Policy Cheat Sheet.” *GitHub*, https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Content_Security_Policy_Cheat_Sheet.md Accessed 17 April 2023.

[16] OWASP. “Denial of Service Cheat Sheet.” *GitHub*, https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Denial_of_Service_Cheat_Sheet.md Accessed 20 April 2023.

[17] OWASP. "SQL Injection Prevention Cheat Sheet." *GitHub*, https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.md#least-privilege Accessed 16 April 2023.

[18] OWASP. "2021 Top 10." *A03 Injection*, https://owasp.org/Top10/ Accessed 15 April 2023.

[19] S, Kirsten. "Cross Site Request Forgery (CSRF)." *OWASP Foundation*, https://owasp.org/www-community/attacks/csrf Accessed 23 April 2023.

[20] Sangwan, Somdev. "XSStrike." *GitHub*, https://github.com/s0md3v/XSStrike Accessed 14 April 2023.

[21] Shrivastava, Ankit, et al. *XSS vulnerability assessment and prevention in web application*. Edited by Amit Agarwal, IEEE, 2016, https://ieeexplore.ieee.org/document/7877529 Accessed 24 April 2023.

[22] Siddiqui, Mohd Shadab, and Deepanker Verma. *Cross site request forgery: A common web application weakness*. IEEE, 2011, https://ieeexplore.ieee.org/document/6014783 Accessed 23 April 2023.

[23] Soewito, Benfano, et al. "Prevention Structured Query Language Injection Using Regular Expression and Escape String." *Science Direct*, 29 July 2022, https://www.sciencedirect.com/science/article/pii/S1877050918315114 Accessed 17 April 2023.

[24] Stasinopoulos, Anastasios, et al. *Bypassing XSS Auditor: Taking advantage of badly written PHP code*, IEEE, 2014, https://ieeexplore.ieee.org/document/7300602 Accessed 12 April 2023.

[25] Steele, Tom, et al., "Building the SQL Injection Fuzzer." *Black Hat Go: Go Programming For Hackers and Pentesters*, No Starch Press, 2020, pp. 193 - 196.

[26] Tanwar, Manish. "Magento eCommerce - Remote Code Execution - XML webapps Exploit." *Exploit-DB*, 26 August 2015, https://www.exploit-db.com/exploits/37977 Accessed 17 April 2023.

[27] Yadav, Pratibha, and Chandresh D. Parekh. *A report on CSRF security challenges & prevention techniques*. IEEE, 2017, https://ieeexplore.ieee.org/document/8275852 Accessed 24 April 2023.

[28] Yaworski, Peter. *Real-World Bug Hunting: A Field Guide to Web Hacking*. No Starch Press, 2019.

[29] Zhang, Boyang, et al. *DDoS detection and prevention based on artificial intelligence techniques*. IEEE, 2017, https://ieeexplore.ieee.org/document/8322748 Accessed 17 April 2023.