South Asian Journal of Engineering and Technology Research Article

Open Access Full Text Article 132(2023) 1-7 DOI: 10.26524/sajet.2023.13.9

IDENTIFYING AND MITIGATING COMMON WEB APPLICATION VULNERABILITIES

Dinesh Gopal Dommeti[1], Persis Voola[2*]

[1]BTech CSE V Semester, Adikavi Nannaya University, College of Engineering, Rajamahendravaram, AP, India.

[2*]Associate Professor, Adikavi Nannaya University, College of Engineering, Rajamahendravaram, AP, India.

Corresponding author.
Correspondence: Persis Voola
E-mail: persisvoola.cse@aknu.edu.in

Article info
Received 28th April 2023 Received in revised form 16 June 2023
Accepted 19 August 2023

Keywords
Web Application Security, Vulnerabilities, SQL Injection, Cross-Site Scripting, Authentication, Burp Suite

https://sajet.in/index.php/journal/article/view/287

Abstract
Web application vulnerabilities involve flaws or weaknesses in web-based applications. This study employs Damn Vulnerable Web Application (DVWA) and Portswigger as testbeds, using the Burp Suite proxy tool to intercept and analyse web traffic. The aim is to identify and exploit common vulnerabilities to enhance application security. Key vulnerabilities explored include Authentication issues, SQL injection, Cross-site scripting (XSS), and Cross-site request forgery (CSRF). The study offers insights into preventing such exploits by securing applications before real-world attacks occur.

1. Introduction

Web applications are integral to modern internet services, but they are often targeted by attackers exploiting various vulnerabilities. This study systematically explores common web application vulnerabilities using DVWA and Burp Suite. Despite the growing awareness of vulnerability databases in the software engineering research community, no comprehensive literature survey has been conducted on their application in software development [1]. The objective is to demonstrate practical exploitation techniques and provide recommendations for securing web applications.

The increasing reliance on web applications for critical services has made them prime targets for cyber-attacks. These applications, while offering convenience and efficiency, often harbour security flaws that can be exploited by malicious actors. Vulnerabilities such as SQL injection, Cross-site scripting (XSS), and Cross-site request forgery (CSRF) can lead to unauthorized access, data breaches, and other significant security incidents. Effective vulnerability assessment and mitigation strategies are essential to safeguard these applications from potential threats.

Key vulnerabilities and case studies are presented, encompassing authentication issues, SQL injection, Cross-site scripting (XSS), and Cross-site request forgery (CSRF). Each case study includes an analysis of practical exploitation examples and offers strategies for prevention. Additionally, recommendations for integrating advanced detection techniques, continuous security assessments, and the use of automated security tools are provided to proactively address and mitigate emerging threats.

2. Key Vulnerabilities and Case Studies

2.1.1 Authentication

Authentication is the process of verifying the identity of users. Weak authentication mechanisms and poor implementation can lead to vulnerabilities, allowing attackers to bypass security. Multi-factor authentication (MFA) enhances security but can be flawed if not implemented correctly.

2.1.2 Case Study: Flawed Two-Factor Verification Logic

An attacker could log in using their credentials but change the value of the account cookie to any arbitrary username when submitting the verification code. This allows access to arbitrary user accounts without knowing their passwords.

2.1.3 Exploit Execution

- Create an account on the vulnerable website.
- Log in to the account and investigate the 2FA verification process.
- Change the verify parameter in the request to the victim's username.
- Submit an invalid 2FA code and brute-force the verification code to gain access.

2.1.4 Analysis and Prevention

Authorizing access to the public cloud has evolved from simple user and password authentication to two-factor authentication (TOTP), which adds a unique code entry, and while multi-factor authentication (MFA) is considered a robust security measure, its effectiveness relies on the proper implementation of each factor; flaws in the second factor's validation logic can allow attackers to bypass it, making MFA ineffective due to common issues such as predictable or reusable verification codes, insecure transmission of codes, and improper session handling after verification, thus highlighting the need for an in-depth examination of implementation to identify weaknesses and guide developers toward more secure practices [6].

2.1.5 Prevention Strategies:

- Use Time-based One-Time Passwords (TOTP): Implement TOTP algorithms that generate a unique code for each authentication attempt.
- Secure Transmission: Ensure that all authentication data is transmitted over secure channels (e.g., HTTPS).
- Session Management: Properly manage user sessions, invalidating old sessions after MFA is completed.
- Brute-force Protection: Implement mechanisms to detect and block brute-force attacks on MFA codes.

2.2 SQL Injection (SQLi)

SQL injection is a code injection technique that exploits software vulnerabilities by manipulating SQL queries, allowing attackers to execute arbitrary SQL code [3].

2.2.1 Case Study: Error-Based SQL Injection

Attackers can retrieve detailed database error messages, which provide information about the database structure. This enables the attacker to craft precise queries to exploit the database further.

2.2.2 Exploit Execution

- Identify a vulnerable input field that interacts with the database.
- Inject SQL code to manipulate the query and retrieve database information.
- Use the extracted information to craft further attacks.

2.2.3 Analysis and Prevention

SQL injection attacks exploit the interaction between web applications and databases. They typically involve inserting or "injecting" malicious SQL code into input fields that are then executed by the backend database. This can lead to unauthorized access to data, modification of data, and even the execution of administrative operations on the database. Advanced SQL injection techniques include blind SQL injection, where attackers infer information through the web application's responses without receiving direct feedback, and second-order SQL injection, which exploits stored data that is later processed insecurely.

2.2.4 Prevention Strategies:

- Parameterized Queries: Use parameterized queries (prepared statements) to separate SQL code from data.
- Stored Procedures: Implement stored procedures to encapsulate SQL logic.
- Input Validation: Validate and sanitize all user inputs to ensure they conform to expected formats.
- Least Privilege Principle: Restrict database user privileges to only those necessary for the application to function.
- Error Handling: Avoid displaying detailed database error messages to users.

2.3 Cross-Site Scripting (XSS)

Cross-site scripting (XSS) attacks involve injecting malicious scripts into web pages, often due to improper sanitization of user inputs, which can steal session cookies, deface websites, or redirect users to malicious sites, causing problems for both users and server applications [4].

2.3.1 Case Study: Reflected XSS

Reflected XSS occurs when user input is immediately returned by the application without proper validation or encoding.

2.3.2 Exploit Execution

- Identify a vulnerable input field.
- Inject malicious JavaScript code.
- The script executes in the user's browser, performing actions on behalf of the user.

2.3.3 Analysis and Prevention

Reflected XSS is one of the most common forms of XSS attacks. It typically occurs when data provided by a web client is used immediately by server-side scripts to generate a web page without properly sanitizing the input. This can lead to the execution of arbitrary JavaScript code in the context of the user's session. To prevent XSS, developers should employ input validation, output encoding, and use security libraries that automatically sanitize inputs. Content Security Policy (CSP) can also be implemented to restrict the sources from which scripts can be executed.

2.3.4 Prevention Strategies:

- Input Validation: Validate all user inputs to ensure they conform to expected formats.
- Output Encoding: Encode data before rendering it in the browser to prevent script execution.
- Security Libraries: Use libraries and frameworks that automatically handle input sanitization and output encoding.
- CSP: Implement Content Security Policy to restrict the sources of executable scripts.
- Regular Audits: Conduct regular security audits and code reviews to identify and fix XSS vulnerabilities.

2.4 Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) is a significant web exploit that tricks victims into submitting malicious requests by inheriting their identity and privileges, thereby performing undesired functions on their behalf, and it continues to pose security risks even on highly ranked websites [5].

2.4.1 Case Study: CSRF Attack on Banking Application

An attacker can craft a request to transfer funds from the victim’s account to their own. When the victim, who is logged into their banking account, unknowingly clicks the link or visits a page containing the malicious request, the transaction is completed.

2.4.2 Exploit Execution

- Craft a malicious request to transfer funds.
- Embed the request in an image tag or hidden form on a website.
- The victim, while logged into their banking account, clicks the link or visits the page, causing the transfer to occur without their consent.

2.4.3 Analysis and Prevention

CSRF attacks exploit the trust that a web application has in the user's browser. When a user is authenticated, the browser automatically includes authentication information (like cookies) with each request. Attackers leverage this behavior by tricking users into making unwanted requests. To prevent CSRF attacks, developers should implement anti-CSRF tokens, which are unique to each session and request, making it difficult for attackers to predict. Additionally, requiring re-authentication for sensitive operations and employing same-site cookie attributes can enhance security.

2.4.4 Prevention Strategies:

- Anti-CSRF Tokens: Implement anti-CSRF tokens that are unique for each session and each request.
- Same-Site Cookies: Use same-site cookie attributes to prevent cookies from being sent with Cross-site requests.
- Re-authentication: Require users to re-authenticate for sensitive operations.
- User Interaction Verification: Verify that requests originate from trusted user interactions, such as by checking referrer headers.
- Security Libraries: Utilize security frameworks and libraries that include built-in CSRF protection mechanisms.

3. Conclusion

Understanding and mitigating web application vulnerabilities is crucial for maintaining security. This study highlights common vulnerabilities, providing practical exploitation examples and defensive strategies. Future work should focus on advanced detection techniques and continuous security assessments to stay ahead of emerging threats. The integration of automated security tools and regular security audits can help in identifying and rectifying vulnerabilities before they are exploited by malicious actors.

4. Further Research

As web applications evolve, so do the techniques employed by attackers. To stay ahead of these threats, future work should focus on integrating advanced technologies, particularly artificial intelligence (AI) and machine learning (ML), to enhance the security posture of web applications. These technologies can provide dynamic, real-time protection by identifying and mitigating vulnerabilities more effectively than traditional methods.

4.1 Automated Vulnerability Detection and Response

AI and ML can be used to develop systems that automatically detect and respond to vulnerabilities. By analysing patterns in web traffic and application behaviour, these systems can identify anomalies indicative of potential attacks. Machine learning algorithms can be trained on vast datasets of known vulnerabilities and attacks, enabling them to recognize and respond to new, previously unseen threats.

Example:

A machine learning model could be trained to detect SQL injection attempts by analysing query patterns. Once a potential attack is identified, the system could automatically block the request and alert security personnel.

4.2 Behavioural Analysis and Anomaly Detection

Anomaly detection models, powered by AI, establish a baseline of normal user behaviour and flag deviations—such as unusual login locations or atypical transaction patterns—triggering alerts and preventive measures, and proving especially effective against sophisticated attacks beyond traditional signature-based detection [2].

Example:

An AI system could detect a brute-force attack by identifying an unusually high number of login attempts from a single IP address within a short period. The system could then automatically implement rate-limiting or temporary IP blocking.

4.3 Enhanced Authentication Mechanisms

AI can improve authentication processes by implementing adaptive authentication. This method evaluates the risk level of each login attempt based on various factors, such as the user’s behaviour, device, and location. High-risk attempts can be challenged with additional authentication steps, such as biometric verification.

Example:

If a user attempts to log in from an unfamiliar device or location, the AI system could require additional verification, such as a fingerprint scan or a one-time password sent to the user's mobile device.

4.4 Predictive Analytics

Over the past decade, predictive analysis has become increasingly important in decision support systems, utilizing statistical algorithms, and IT tools to identify dependencies and patterns in data sets while reducing complexity. In essence, it aims to build models that determine the probability of future events based on historical data for new situations [7].

Example:

By analysing past security incidents and current threat intelligence, an AI system could predict which vulnerabilities are most likely to be targeted next and prioritize their remediation.

4.5 Integration with DevSecOps

Incorporating AI into the DevSecOps pipeline ensures that security is integrated throughout the software development lifecycle. AI-driven tools can perform continuous code analysis, identifying and addressing vulnerabilities as the code is being written. This reduces the risk of vulnerabilities making it into production.

Example:

An AI-powered static code analysis tool could automatically scan code for common vulnerabilities, such as buffer overflows or improper input validation, and provide developers with real-time feedback and recommendations.

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to everyone who assisted in the completion of this research paper. I am fortunate to pursue my Bachelor's Degree at Adikavi Nannaya University, which provided excellent facilities in Computer Science and Engineering. Special thanks to Dr. V. Persis, Associate Professor & Principal, and the Head of the Department for their encouragement and guidance. I am also grateful to the management for the opportunity to work on "Identifying and Mitigating Common Web Application Vulnerabilities." Lastly, I appreciate the support from the review committee and faculty members throughout this research.

Funding

No funding was received to carry out this study.

References

1. S. S. Alqahtani, E. E. Eghan, & J. Rilling, “SV-AF—A Security Vulnerability Analysis Framework”, IEEE, 27th International Symposium on Software Reliability Engineering (ISSRE), pp. 219-229, 2016.
2. K. G. Maheswari, & R. Anita, “An Intelligent Detection System for SQL Attacks on Web IDS in a Real-Time Application” Springer, Proc. of the 3rd International Symposium on Big Data and Cloud Computing Challenges (ISBCC–16’,) pp. 93-99, 2016.
3. R. V. Bhor, & H. K. Khanuja, “Analysis of web application security mechanism and Attack Detection using Vulnerability injection technique”, IEEE, International Conference on Computing Communication Control and automation (ICCUBEA), pp. 1-6, 2016.
4. I. Hydara, A.B.M. Sultan, H. Zulzalil, N. Admodisastro, “Current state of research on Cross-site scripting (XSS)” – A systematic literature review Information and Software Technology, 58 (2015), pp. 170-186.
5. Nuel Siahaan , Mario Rufisanto , Raymond Nolasco , Said Achmad , Chrisando Ryan Pardomuan Siahaan, “Study of Cross-Site Request Forgery on Web-Based Application: Exploitations and Preventions” pp.1-7, 2023.
6. I. Gordin, A. Graur, A. Potorac, “Two-factor authentication framework for private cloud” 2019 23rd International Conference on System Theory, Control and Computing (ICSTCC) (2019), pp. 255-259.
7. Maciej Wach , Iwona Chomiak-Orsa, ” The application of predictive analysis in decision-making processes on the example of mining company’s investment projects”, pp. 5058-5066, 2021.