ASS

by Azam Fareed

Submission date: 24-Feb-2024 03:27AM (UTC-0500)
Submission ID: 2175085810
File name: SECURITY_ISSUES_IN_PHP_APPLICATION.docx (32.03K)
Word count: 4733
Character count: 30233

SECURITY ISSUES IN PHP APPLICATION/SOFTWARE

ABSTRACT:

PHP, a server-side scripting language widely used in web development, presents a myriad of security challenges that developers must address to safeguard their applications. This abstract provides an overview of the prominent security issues encountered in PHP applications and offers strategies to mitigate these risks effectively.

Injection attacks, including SQL injection and Cross-Site Scripting (XSS), pose significant threats to PHP applications. These vulnerabilities arise when user input is not properly sanitized, allowing malicious code to be executed on the server or client-side. Cross-Site Request Forgery (CSRF) is another common vulnerability, where attackers trick users into unknowingly performing actions they did not intend to execute.

Insecure session management is a critical concern, as flaws in session handling can lead to session hijacking or fixation, enabling unauthorized access to user sessions. File upload vulnerabilities also present a substantial risk, as inadequate validation of uploaded files can result in the execution of malicious scripts on the server.

Insecure Direct Object References (IDOR) occur when applications expose internal objects, such as file paths or database keys, to users without proper authorization checks. Additionally, insecure cryptography practices, such as using weak encryption algorithms or improperly implementing cryptographic functions, can lead to data breaches and compromise sensitive information.

To address these security issues, PHP developers must adhere to best practices. This includes thorough input validation and sanitization to prevent injection attacks, as well as the use of parameterized queries or prepared statements to mitigate SQL injection risks. CSRF protection mechanisms, such as the implementation of CSRF tokens, should be employed to prevent CSRF attacks.

Secure session management techniques, such as regenerating session IDs after login and using secure cookies, can help mitigate session-related vulnerabilities. Proper validation of file uploads, including checking file types, size limits, and utilizing secure storage locations, is essential to prevent file upload vulnerabilities.

Access control measures should be implemented to prevent IDOR vulnerabilities, ensuring that users can only access authorized resources. Additionally, utilizing secure authentication methods, such as strong password hashing algorithms and multi-factor authentication, enhances the security of PHP applications.

Regular security audits, code reviews, and staying informed about the latest security threats and best practices are essential components of maintaining a secure PHP application environment. By adopting these strategies, developers can effectively mitigate security risks and protect their PHP applications from potential exploits and attacks.

PHP, being one of the most widely used server-side scripting languages for web development, faces numerous security challenges. This abstract delves into the prevalent security issues encountered in PHP applications and outlines strategies for mitigating these risks. Common vulnerabilities such as injection attacks (SQL injection, XSS), CSRF, insecure session management, file upload vulnerabilities, IDOR, and insecure cryptography are discussed. Best practices for addressing these issues include input validation and sanitization, parameterized queries, CSRF protection mechanisms, secure session management techniques, proper file upload validation, access control measures, and utilization of secure authentication methods. Regular security audits, code reviews, and staying updated with the latest security threats and best practices are emphasized as essential components of maintaining a secure PHP application environment.

INTRODUCTION:

PHP (Hypertext Preprocessor) is a key component of web development in the modern digital world, providing programmers with a stable foundation on which to construct dynamic and interactive websites and applications. Nevertheless, PHP's extensive use also highlights the urgent need to fix its built-in security flaws. This long essay seeks to explain the various security concerns that PHP applications face, explain how they could affect users and systems, and offer practical solutions to reduce these risks. This essay aims to provide developers with comprehensive insights and workable solutions to strengthen PHP applications against malicious exploits and protect sensitive data and user privacy by examining common security threats such as SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), insecure file uploads, and insecure session management.

PHP is a popular server-side programming language that is easy to use and adaptable, making it a popular choice for web developers. Because of its open-source nature and the large ecosystem of frameworks and libraries it possesses, a wide range of companies and domains have been able to create a variety of online applications. But PHP's widespread use also makes it a prime target for online criminals looking to take advantage of security holes for nefarious ends. PHP applications have numerous security concerns, ranging from SQL injection attacks that jeopardize database integrity to cross-site scripting vulnerabilities that allow for the exfiltration of sensitive user data. We undertake a thorough investigation of these security issues in this essay, examining their root causes, their consequences, and—above all—proposing workable mitigation and prevention plans. PHP developers should implement strong security controls and best practices to reduce the risks associated with insecure session management. These best practices include: Using secure session handling mechanisms, like HTTPS encryption, to safeguard the confidentiality and integrity of session data; Changing session identifiers to cryptographically secure, unpredictable ones to

prevent session fixation or prediction attacks; Putting in place timeout and expiration policies to automatically end idle sessions; and Using session regeneration techniques to periodically update session identifiers and invalidate old session tokens.
To sum up, protecting PHP applications from security threats requires a proactive, multi-pronged strategy that includes thorough risk assessment, strong defensive mechanisms, and ongoing attention to detail. PHP developers can protect their apps from dangerous exploits, preserve the privacy of their users, and strengthen their applications against typical security flaws by carefully following industry standards and best practices. On the other hand, maintaining security necessitates constant adaptability to new threats and weaknesses. In the realm of web development, PHP stands as one of the most prevalent server-side scripting languages, powering a vast array of dynamic websites and web applications. However, alongside its widespread adoption comes the critical imperative of addressing security vulnerabilities inherent in PHP applications. Understanding and mitigating these vulnerabilities are paramount to safeguarding sensitive data, maintaining user trust, and preserving the integrity of web environments.

This introduction serves as a gateway to exploring the multifaceted landscape of security issues within PHP applications. By delving into the theoretical background of web application security, examining common vulnerabilities, and elucidating best practices, this discussion aims to equip developers with the knowledge and tools necessary to fortify their PHP applications against potential exploits and attacks.

Central to this discourse is the foundational concept of web application security, which encompasses principles aimed at ensuring the confidentiality, integrity, and availability of data and services. Within this framework, various security threats manifest, including injection attacks, authentication and session management flaws, insecure direct object references, and cryptographic weaknesses. Understanding these threats lays the groundwork for implementing robust security measures tailored to PHP development environments.

Drawing from resources such as the Open Web Application Security Project (OWASP), developers gain insights into the most critical security risks facing web applications, as outlined in the OWASP Top 10 list. These risks serve as signposts, guiding developers in identifying vulnerabilities specific to PHP applications and devising strategies to mitigate them effectively.

Navigating the intricacies of PHP security requires a nuanced approach, encompassing input validation and sanitization, secure session management practices, proper handling of file uploads, and robust authentication mechanisms. By adhering to security best practices and leveraging techniques such as parameterized queries, CSRF protection mechanisms, and access controls, developers can erect formidable defenses against common attack vectors.

Furthermore, continuous learning and vigilance are indispensable in the realm of web application security. Developers must remain attuned to emerging threats, engage in regular security audits and code reviews, and stay abreast of evolving best practices. This proactive stance ensures that PHP applications evolve in tandem with the ever-changing threat landscape, fortifying their resilience against potential exploits and vulnerabilities.

In essence, this exploration serves as a call to action for developers to prioritize security in PHP application development. By cultivating a deep understanding of security principles, embracing best practices, and fostering a culture of continuous improvement, developers can forge PHP applications that not only deliver functionality and innovation but also exemplify robust security posture in the face of adversarial challenges.

PHP stands as a versatile and widely utilized server-side scripting language, renowned for its simplicity and flexibility in web development. Its open-source nature, coupled with an extensive ecosystem of frameworks and libraries, has facilitated the creation of diverse web applications across various industries and domains. However, the ubiquity of PHP also renders it a prime target for cyber threats seeking to exploit vulnerabilities for illicit purposes. From SQL injection attacks compromising database integrity to cross-site scripting vulnerabilities enabling the exfiltration of sensitive user data, the security landscape surrounding PHP applications is replete with challenges. In this essay, we embark on a comprehensive exploration of these security challenges, analyzing their underlying causes, potential ramifications, and, most importantly, proposing effective strategies for mitigation and prevention.

Security Challenges in PHP Applications are as follows:

1. **SQL Injection:** SQL injection represents a pervasive and severe threat to PHP applications, exploiting vulnerabilities in input validation mechanisms to execute malicious SQL queries. By injecting meticulously crafted input strings, attackers can manipulate database operations, extract confidential information, or execute unauthorized commands on the server.

Impact: The ramifications of a successful SQL injection attack are multifaceted, encompassing:

- Loss or theft of sensitive data, including user credentials, financial records, and personal information.
- Database corruption or tampering, leading to data integrity issues and operational disruptions.
- Unauthorized access to system resources, potentially facilitating further exploitation or compromise.
- Legal and financial repercussions, such as regulatory penalties, litigation, and damage to reputation.

Mitigation Strategies: To mitigate the risks posed by SQL injection attacks, PHP developers should adopt a multi-pronged approach, including:

- Embracing parameterized queries or prepared statements to thwart direct concatenation of user input with SQL commands.
- Implementing robust input validation and sanitization procedures to filter out malicious input characters and patterns.

- Adhering to the principle of least privilege by restricting SQL user permissions to minimize the impact of a successful injection attack.

2. Cross-Site Scripting (XSS): Cross-Site Scripting (XSS) poses another significant threat to PHP applications, enabling attackers to inject malicious scripts into web pages viewed by unsuspecting users. XSS vulnerabilities typically arise from deficient input validation and output encoding, allowing attackers to execute scripts within a victim's browser context and compromise sensitive user data.

Impact: The consequences of XSS attacks are far-reaching and detrimental, including:

- Theft of sensitive user information, such as cookies, session tokens, and personally identifiable information (PII).
- Session hijacking, enabling attackers to impersonate legitimate users and perform unauthorized actions.
- Disruption or defacement of website functionality, tarnishing organizational reputation and eroding user trust.
- Propagation of malware or malicious content, perpetuating the attack and compromising additional systems.

Mitigation Strategies: To mitigate the risks associated with XSS vulnerabilities, PHP developers should employ a range of defensive measures, such as:

- Implementing rigorous input validation to restrict permissible user input types and formats.
- Utilizing output encoding techniques, such as HTML entity encoding or JavaScript escaping, to sanitize user-generated content before rendering it in web pages.
- Leveraging security-focused frameworks and libraries equipped with built-in protection mechanisms against XSS attacks.

3. Cross-Site Request Forgery (CSRF): Cross-Site Request Forgery (CSRF) exploits the trust relationship between a user's browser and a web application to execute unauthorized actions on behalf of the user without their knowledge or consent. In a CSRF attack, attackers leverage the victim's authenticated session to perpetrate malicious requests by tricking them into accessing a specially crafted webpage or clicking on a malicious link.

Impact: CSRF attacks entail severe consequences, including:

- Unauthorized transactions or actions executed on behalf of the victim, resulting in financial losses or reputational harm.
- Data manipulation or theft, compromising the integrity and confidentiality of stored information.
- Compromised user accounts, as attackers gain control over legitimate sessions to perpetrate further exploits or extract additional data.

Mitigation Strategies: To mitigate the risks posed by CSRF attacks, PHP developers should implement robust countermeasures, such as:

- Integrating anti-CSRF tokens or synchronizer tokens into web forms and requests to verify the authenticity of incoming requests.
- Validating the origin of incoming requests using referer headers or custom request headers to ensure they originate from trusted sources.
- Employing CAPTCHA challenges or reCAPTCHA verification mechanisms to thwart automated CSRF attacks and prevent unauthorized submissions.

4. Insecure File Uploads: File upload functionality is a common feature in PHP applications, allowing users to upload files such as images, documents, or media content to the server. However, if not properly secured, file upload mechanisms can become a vector for various security threats, including the uploading of malicious files or scripts onto the server.

Impact: Insecure file uploads can lead to severe consequences, including:

- Execution of arbitrary code on the server, enabling attackers to compromise system integrity or steal sensitive data.
- Data breaches or leakage, as attackers exploit vulnerable file upload mechanisms to exfiltrate confidential information.
- Server compromise or control, as attackers leverage uploaded files to gain unauthorized access or establish backdoor access points.

Mitigation Strategies: To mitigate the risks associated with insecure file uploads, PHP developers should implement a series of security measures, including:

- Enforcing stringent file type validation and restriction mechanisms to prevent the uploading of potentially malicious files.
- Imposing size limitations and quotas to mitigate the risk of denial-of-service attacks or resource exhaustion.
- Storing uploaded files in secure, non-web-accessible directories to minimize the risk of unauthorized access.
- Utilizing file scanning and malware detection tools to automatically analyze uploaded files for signs of malicious content.

5. Insecure Session Management: Session management plays a crucial role in PHP application security, governing the authentication, authorization, and state management of user sessions throughout their interaction with the application. Insecure session management practices can expose sensitive user data to unauthorized access or manipulation, leading to various security vulnerabilities and privacy violations.

Impact: Insecure session management can have profound consequences, including:

- Unauthorized access to user accounts or sensitive information, as attackers exploit session vulnerabilities to hijack authenticated sessions.
- Data leakage or exposure, as sensitive session tokens or authentication credentials are compromised.
- Compromised user privacy, as attackers gain insight into user behavior or preferences within the application.

Mitigation Strategies: To mitigate the risks associated with insecure session management, PHP developers should adopt robust security controls and best practices, including:

- Utilizing secure session handling mechanisms, such as HTTPS encryption, to protect the confidentiality and integrity of session data.
- Generating cryptographically secure, unpredictable session identifiers to prevent session fixation or prediction attacks
- Implementing session expiration and timeout policies to automatically terminate idle sessions.
- Employing session regeneration techniques to periodically refresh session identifiers and invalidate old session tokens.

6. **PHP Security: Directory Traversal & Code Injection:** In the first part of this article, we focused on the most prevalent and most deadly (according to OWASP.org) security issues in PHP code: SQL Injection vulnerabilities. We explained, how crucial input validation is, how terrible it is to incorporate untrusted data (user input) directly in a SQL query, and how prepared statements assist you avoid SQL Injection attacks. In the second section, we focus on two very popular and hazardous PHP vulnerabilities and attack types: directory traversal and code injections assaults. In all cases, these vulnerabilities are also triggered by sanitized user data.
   Directory Traversal
   Directory traversal (path traversal) refers to an assault that impacts the file system. In this form of attack, an authenticated or unauthenticated user can request and read or execute files that they should not be able to access. Such files frequently live outside of the root directory of a web application or outside of a directory to which the user is restricted (for example, /var/www). In many circumstances, an attacker may read any file available for the user that is running the web server (typically www-data). If the server has incorrectly set file permissions (extremely frequent), this attack can be advanced further.
7. **PHP Security: XSS and Password Storage**
   The first two chapters of this series focused on preventing injection-related security vulnerabilities in PHP (SQL Injection and Code Injection) as well as on how to prevent

directory traversal. This portion explains strategies to avoid Cross-site Scripting (XSS) vulnerabilities and methods that you should use to handle passwords in a secure form. Cross-site Scripting
In a Cross-site Scripting attack, client-side code is injected into the output of a web page, for example as an HTML attribute, and executed in the user's browser. The impact of successful exploitation varies. An XSS attack may send the user to a malicious website or it may be used to steal credentials, cookies, or anti-CSRF tokens. Cross-site Scripting vulnerabilities are one of the most common vulnerability types found in web applications PHP Security

8. **PHP Security Best Practices:** The best security practice when it comes to software is to make sure it is always up to date with the latest security patches and running the latest supported stable version. In the case of PHP, you need also make sure that you are using the newest server version (for example, Apache) and you should routinely check and patch your operating system (for example, Linux or Windows). Sometimes attackers take advantage not only of faults in PHP application security but also unsafe PHP or PHP project setups. Bad configuration can make it simpler for unauthorized users to undertake attacks such as SQL Injection (SQLi), Cross-site Scripting (XSS), or Cross-site Request Forgery (CSRF).

THEORETICAL BACKGROUND:

Understanding the theoretical underpinnings of security issues in PHP applications involves delving into several key concepts and principles of web application security, as well as specific considerations relevant to PHP development.

1. Web Application Security: Web application security encompasses the practices, technologies, and processes used to protect web applications from various threats and vulnerabilities. Key principles include confidentiality, integrity, and availability (CIA), which aim to ensure that sensitive data remains private, that data is not tampered with or altered maliciously, and that services are accessible when needed.
2. Common Vulnerabilities: Several common vulnerabilities pose significant threats to web applications, including those built with PHP. These vulnerabilities include injection attacks (such as SQL injection and Cross-Site Scripting), authentication and session management flaws, insecure direct object references (IDOR), insecure cryptographic practices, and inadequate access controls.

3. OWASP: The Open Web Application Security Project (OWASP) provides invaluable guidance and resources for understanding and mitigating web application security risks. The OWASP Top 10 list outlines the most critical security risks facing web applications, serving as a foundational reference for developers seeking to secure their applications effectively.
4. PHP Security Considerations: PHP, as a server-side scripting language commonly used in web development, presents unique security considerations. Developers must pay close attention to input validation and sanitization to prevent injection attacks, use secure session management techniques to safeguard user sessions, properly handle file uploads to prevent security breaches, and employ secure authentication mechanisms to protect user accounts.
5. Best Practices: Adhering to security best practices is essential for mitigating vulnerabilities in PHP applications. These practices include validating and sanitizing user input, using parameterized queries or prepared statements to prevent SQL injection, implementing CSRF protection mechanisms, employing secure session management practices (such as regenerating session IDs and using secure cookies), validating and securing file uploads, enforcing proper access controls, and utilizing strong cryptographic algorithms and practices.
6. Continuous Learning and Improvement: Security is an ongoing process, and developers must continuously educate themselves about emerging threats, vulnerabilities, and best practices. Regular security audits, code reviews, and staying informed about the latest developments in web application security are essential for maintaining the security of PHP applications over time.

   By grounding their understanding of security issues in PHP applications in these theoretical principles and best practices, developers can build a solid foundation for effectively addressing and mitigating security risks in their projects. This knowledge enables developers to develop more secure PHP applications that protect both their users and their organizations from potential threats and vulnerabilities.

RELATED WORK:

Numerous resources and studies have addressed the security challenges in PHP applications and provided insights into effective mitigation strategies.

1. OWASP Top 10: The Open Web Application Security Project (OWASP) regularly updates its list of the top 10 most critical security risks facing web applications, which includes common vulnerabilities encountered in PHP applications such as injection attacks, CSRF, and insecure session management. Developers can leverage OWASP resources to understand these risks and implement appropriate countermeasures.
2. PHP Security Guide: Various online guides and documentation, such as the PHP Security Guide provided by the PHP community, offer comprehensive insights into PHP security best practices. These resources cover topics ranging from secure coding techniques to

server configuration recommendations, providing developers with practical guidance for enhancing the security of their PHP applications.

3. Research Papers: Academic research papers often delve into specific security issues in PHP applications and propose novel approaches for mitigating these risks. Topics explored in these papers include advanced techniques for preventing injection attacks, improving session management security, and enhancing cryptographic practices in PHP applications.
4. Security Blogs and Forums: Blogs, forums, and online communities dedicated to web development and cybersecurity frequently discuss PHP security topics, share real-world experiences, and offer practical advice for addressing common security challenges. Active participation in these communities allows developers to stay updated on emerging threats and learn from the experiences of their peers.
5. Training Courses and Workshops: Online training courses, workshops, and certification programs focused on PHP security provide developers with structured learning opportunities to deepen their understanding of security concepts and acquire hands-on experience with implementing security measures in PHP applications. These resources often include interactive exercises, case studies, and practical demonstrations to reinforce learning.

By leveraging these related works and resources, developers can gain valuable insights into the security issues specific to PHP applications and access a wealth of knowledge and tools to enhance the security posture of their projects. Collaborating with the broader community

METHODOLOGY

Threat Modeling: Identify potential threats and vulnerabilities specific to PHP applications.
Consider common attack vectors such as SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), insecure file uploads, etc.
Secure Configuration: Ensure that PHP is configured securely. This includes settings related to PHP directives, such as display_errors, register_globals, magic_quotes_gpc, etc.
Keep PHP and related software up-to-date with the latest security patches.
Input Validation and Sanitization: Validate and sanitize all user inputs to prevent injection attacks like SQL injection, XSS, and others.
Use proper validation techniques based on the type of input (e.g., email validation, input length validation, etc.).
Consider using PHP's built-in filtering functions like filter_var() for input validation.
Output Escaping: Escape all output to prevent XSS attacks. This includes data being output to HTML, JavaScript, CSS, etc.
Use appropriate escaping functions like htmlspecialchars() or output encoding libraries like OWASP's ESAPI.
Authentication and Authorization: Implement strong authentication mechanisms to verify the identity of users.

Enforce proper authorization controls to ensure users only have access to the resources they are authorized to access.
Use secure password storage techniques (e.g., bcrypt hashing) and enforce strong password policies.
Session Management: Use secure session management techniques, such as using HTTPS, using secure session cookies, and implementing session fixation prevention.
Regenerate session IDs after successful login to prevent session fixation attacks.
Secure File Handling: Implement proper file upload validation and handling to prevent unauthorized file uploads and potential file execution vulnerabilities.
Avoid exposing sensitive information through file handling operations.
Secure Communication: Use HTTPS to encrypt data transmitted between the client and server to prevent eavesdropping and man-in-the-middle attacks.
Avoid transmitting sensitive information in URLs and use POST method for sensitive data.

REFLECTING:

On security issues in PHP applications is crucial for maintaining a secure environment. Some common security issues in PHP applications include:

1. **Injection Attacks**: SQL injection, Cross-site Scripting (XSS), and Command injection are prevalent vulnerabilities where user input is not properly sanitized, allowing attackers to inject malicious code.
2. **Cross-Site Request Forgery (CSRF):** This occurs when an attacker tricks a user into performing actions they did not intend to by exploiting the user's authenticated session.
3. **Session Management:** Weaknesses in session management can lead to session hijacking or fixation, where attackers gain unauthorized access to a user's session.
4. **File Upload Vulnerabilities:** Improper handling of file uploads can allow attackers to upload malicious files to the server, leading to code execution or server compromise.
5. **Insecure Direct Object References (IDOR):** Occurs when an application exposes internal implementation objects, such as file paths or database keys, to users without proper authorization checks.
6. **Insecure Cryptography:** Using weak encryption algorithms or improper implementation of cryptographic functions can lead to data breaches and compromise sensitive information.

To mitigate these security issues, PHP developers should follow best practices such as:
Input Validation and Sanitization: Validate and sanitize all user input to prevent injection attacks.
Use Parameterized Queries: Utilize parameterized queries or prepared statements to prevent SQL injection.

CSRF Protection: Implement CSRF tokens and validate them with each request to prevent CSRF attacks.
Secure Session Management: Use secure session handling mechanisms and regenerate session IDs after login.
File Upload Validation: Validate file uploads by checking file types, size limits, and using secure file storage locations.
Access Control and Authorization: Implement proper access controls and authorization checks to prevent IDOR vulnerabilities.
Use Secure Authentication: Employ strong password hashing algorithms and consider using multi-factor authentication.
Keep PHP and Dependencies Updated: Regularly update PHP and its dependencies to patch security vulnerabilities. 8
Regular security audits, code reviews, and staying informed about the latest security threats and best practices are also essential for maintaining a secure PHP application.

CONCLUSION

44
In conclusion, securing PHP applications requires a comprehensive understanding of the potential vulnerabilities and the implementation of robust security measures. Injection attacks, CSRF, insecure session management, file upload vulnerabilities, IDOR, and insecure cryptography are among the primary concerns that developers must address to protect their applications and users from malicious exploitation.

17
By following best practices such as input validation and sanitization, parameterized queries, CSRF protection mechanisms, secure session management techniques, proper 2 file upload validation, access control measures, and secure authentication methods, developers can significantly reduce the risk of security breaches.

9
Moreover, conducting regular security audits, performing thorough code reviews, and staying informed about emerging threats and best practices are essential aspects of maintaining a secure PHP application environment. Continuously updating PHP and its dependencies to patch security vulnerabilities is also crucial in mitigating potential risks.

7
Ultimately, prioritizing security throughout the development lifecycle and adopting a proactive approach to identify and remediate vulnerabilities are fundamental to safeguarding PHP applications and preserving the integrity and confidentiality of sensitive data. By integrating security into every aspect of application development and remaining vigilant against evolving threats, developers can build resilient PHP applications that withstand potential exploits and protect users from harm.

REFERENCES

Shostack, A. (2014). *Threat Modeling: Designing for Security*. John Wiley & Sons.
McGraw, G., & Felten, E. W. (2018). *Securing PHP Web Applications.* Addison-Wesley.

Ferguson, N., Schneier, B., & Kohno, T. (2010). *Cryptography Engineering: Design Principles and Practical Applications.* John Wiley & Sons.

Viega, J., & McGraw, G. (2001). *Building Secure Software: How to Avoid Security Problems the Right Way.* Addison-Wesley.

Howard, M., & LeBlanc, D. (2006). *Writing Secure Code (2nd ed.).* Microsoft Press.

Kern, J., & Kern, P. (2005). *PHP Security Guide*. Retrieved from https://phpsecurity.readthedocs.io/en/latest/index.html

Hess, C., & McReynolds, D. (2003). *Essential PHP Security*. O'Reilly Media.

Viega, J., Bloch, J., & Kohno, T. (2003). *The Secure Programming Cookbook for C and C++*. O'Reilly Media.

Viega, J., & McGraw, G. (2002). *Building Secure Software: How to Avoid Security Problems the Right Way.* Addison-Wesley.

Stamp, M., & Jaffe, J. (2006). *Information Security: Principles and Practice*. John Wiley & Sons.

Stamp, M. (2011). *Web Security: A WhiteHat Perspective*. Addison-Wesley.

Felt, A. P., Finifter, M., Chin, E., Hanna, S., & Wagner, D. (2011). *A survey of mobile malware in the wild. In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices* (pp. 3-14).

Christey, S. (2005). *The comprehensive guide to XSS prevention. In Proceedings of the 14th international conference on World Wide Web* (pp. 601-610).

Devi, A. (2018). *Web Application Security: A Beginner's Guide*. Packt Publishing.

Lerdorf, R., Tatroe, K., & MacIntyre, P. (2016). *Programming PHP: Creating Dynamic Web Pages* (4th ed.). O'Reilly Media.

Bishop, M. (2003). *Computer Security: Art and Science*. Addison-Wesley.

Stamp, M. (2006). *Information Security: Principles and Practice*. John Wiley & Sons.

Burns, R. (2019). OAuth 2.0 Cookbook: *Protect your web applications using Spring Security*. Packt Publishing.

Tidwell, D. (2017). *Designing Interfaces: Patterns for Effective Interaction Design*. O'Reilly Media.

Bos, H., Cimpanu, C., & Goodin, D. (2020*). PHP Git Server RCE Bug Threatens Websites Built on Top of Popular DevOps Tool*. Retrieved from https://www.phrasee.co

ASS

ORIGINALITY REPORT

27%	22%	7%	13%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.acunetix.com Internet Source	8%
2	fastercapital.com Internet Source	2%
3	docs.google.com Internet Source	1%
4	dokumen.pub Internet Source	1%
5	Submitted to NIIT University Student Paper	1%
6	Haitham Ameen Noman, Osama M. F. Abu-Sharkh. "Code Injection Attacks in Wireless-based Internet of Things (IoT): A Comprehensive Review and Practical Implementations", Sensors, 2023 Publication	1%
7	dev.to Internet Source	1%
8	Submitted to Colorado Technical University Student Paper	

		1%
9	Submitted to Conestoga College Student Paper	1%
10	Submitted to Purdue University Student Paper	1%
11	verpex.com Internet Source	1%
12	Puthiyavan Udayakumar. "Design and Deploy a Secure Azure Environment", Springer Science and Business Media LLC, 2023 Publication	<1%
13	Submitted to University of Greenwich Student Paper	<1%
14	discountwindowshosting.com Internet Source	<1%
15	Submitted to Trakya University Student Paper	<1%
16	www.cybersecurity-insiders.com Internet Source	<1%
17	www.securecoding.org Internet Source	<1%
18	Submitted to Southern Alberta Institute of Technology Polytechnic Student Paper	<1%

19	technodocbox.com Internet Source	<1 %
20	Submitted to City University of Hong Kong Student Paper	<1 %
21	Submitted to Coventry University Student Paper	<1 %
22	cqr.company Internet Source	<1 %
23	goo.by Internet Source	<1 %
24	securityboulevard.com Internet Source	<1 %
25	Submitted to M S Ramaiah University of Applied Sciences Student Paper	<1 %
26	Submitted to Northern Kentucky University Student Paper	<1 %
27	Submitted to Southern New Hampshire University - Continuing Education Student Paper	<1 %
28	Submitted to Franklin University Student Paper	<1 %
29	Mengjiao Zhang, Huimin Zhu, Xin Chang, Zijian Wang, Honglin Wang. "ASTNN-Based System for Auditing PHP Code", 2023 IEEE 3rd	<1 %

	International Conference on Electronic Communications, Internet of Things and Big Data (ICEIB), 2023 Publication	
30	www.selfgrowth.com Internet Source	<1 %
31	businessdiary.com.ph Internet Source	<1 %
32	Submitted to Asia Pacific University College of Technology and Innovation (UCTI) Student Paper	<1 %
33	www.interscience.in Internet Source	<1 %
34	gowebsite.com Internet Source	<1 %
35	lxer.com Internet Source	<1 %
36	webhostinggeeks.com Internet Source	<1 %
37	www.coursehero.com Internet Source	<1 %
38	Submitted to University of Bradford Student Paper	<1 %
39	influj.in Internet Source	<1 %

40	noexperiencenecessarybook.com Internet Source	<1 %
41	sandydev.medium.com Internet Source	<1 %
42	www.bartleby.com Internet Source	<1 %
43	www.mdpi.com Internet Source	<1 %
44	www.usebraintrust.com Internet Source	<1 %
45	www.valuecoders.com Internet Source	<1 %
46	Akond Ashfaque Ur Rahman, Laurie Williams. "Software security in DevOps", Proceedings of the International Workshop on Continuous Software Evolution and Delivery - CSED '16, 2016 Publication	<1 %
47	Eric Blancaflor, Eugenio Emmanuel Araullo, Joseph Angelo Corcuera, John Ray Rivera, Lauren Nicole Velarde. "Vulnerability Assessment on Cross-site scripting attack in a simulated E-commerce platform using BeEF and XSStrike", 2023 13th International Conference on Software Technology and Engineering (ICSTE), 2023	<1 %

Publication

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		