Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2022, 9(6):108-113**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Input Validation: Validating User Inputs to Prevent Injection Attacks and Data Manipulation

Naga Satya Praveen Kumar Yadati

Email: praveenyadati@gmail.com

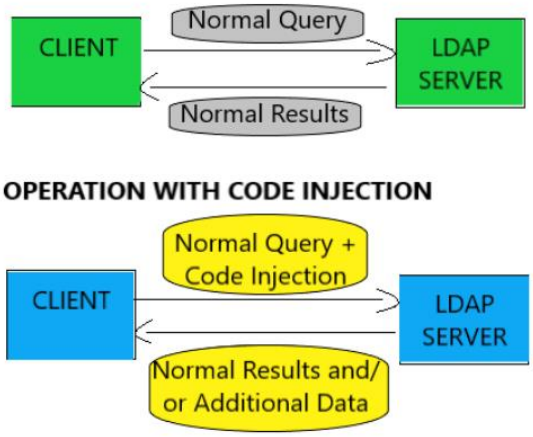
Abstract In recent years, a significant increase in attacks and data breaches has been observed, with a large portion targeting web application vulnerabilities. Mitigating these vulnerabilities has become a crucial research area. Due to the potentially severe impacts of web application vulnerabilities, various approaches have been proposed over the past decades to mitigate their effects. Among these, input validation vulnerabilities represent a critical concern as they arise from unvalidated external data processed by web applications. This paper presents a systematic review of input validation vulnerabilities, including a new classification system and an evaluation of various detection techniques and tools. By examining the strengths and weaknesses of these methods, this paper aims to provide comprehensive countermeasures to enhance web security.

Keywords Input Validation, Injection Attacks, Data Manipulation

1. Introduction

Since the advent of the World Wide Web (WWW) in the early 1990s, web applications have become ubiquitous, hosting everything from simple static text pages to complex applications such as online banking, social media, and e-commerce platforms. However, the continuous introduction of new technologies and the increasing complexity of web applications have led to the emergence of numerous vulnerabilities. Reports from organizations like OWASP and SANS frequently highlight new and evolving web vulnerabilities, emphasizing the need for effective mitigation strategies.

This paper focuses on input validation issues that occur in cloud-based or in-house applications due to insufficient validation of user inputs. These vulnerabilities can lead to severe consequences, such as SQL injection, cross-site scripting (XSS), and other injection attacks. The goal of this paper is to provide a systematic review of input validation vulnerabilities, categorize them, and evaluate various detection and prevention techniques.



2. Background and Motivation

Web application vulnerabilities often arise from improper handling of external data received through user inputs, APIs, or other integration points. For instance, in PHP, user inputs received via the $_GET method may be directly processed by sensitive functions like mysqli_query without proper validation, making the application vulnerable to SQL injection attacks. Figure 1 illustrates the concept of source (external data input) and sink (sensitive function) in source code review.

Example: SQL Injection in PHP

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "database";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$userInput = $_GET['userInput'];
$sql = "SELECT * FROM users WHERE username = '$userInput'";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " . $row["username"]. "<br>";
    }
} else {
    echo "0 results";
}
$conn->close();
?>
```

In this example, the $_GET['userInput'] parameter is directly included in the SQL query without validation, making the application vulnerable to SQL injection attacks.

The motivation for this paper stems from the observation that while many studies discuss web vulnerabilities in general, few focus specifically on input validation issues. These vulnerabilities are particularly problematic because they can be easily exploited by malicious inputs if not properly sanitized. Existing reviews often suffer from poor categorization and overlap, leaving gaps in the literature. This paper aims to address these gaps by providing a detailed classification and comprehensive review of input validation vulnerabilities and their countermeasures.

3. Methodology

Our methodology involves data collection, vulnerability analysis, and classification. We reviewed a total of 720 papers published between 2015 and 2022, using keywords related to web security, input validation vulnerabilities, and various types of injections (e.g., SQL, XSS, XPath). The papers were sourced from

reputable databases such as IEEE, ACM, Springer, and OWASP. The selected research primarily focuses on SQL, XPATH, and XSS injection vulnerabilities, which are prevalent in modern web applications.

4. Techniques for Reducing Web Application Vulnerabilities

There are two main techniques for mitigating web application vulnerabilities: static analysis and dynamic analysis.

1. **Static Analysis:** This technique involves examining the source code without executing the program. Tools like Fortify, Checkmarx, and SonarQube analyze the code to identify potential vulnerabilities early in the development cycle. Static analysis can be highly effective in detecting input validation issues before the application is deployed.
2. **Dynamic Analysis:** This technique involves testing the application during runtime to identify vulnerabilities. Tools like Burp Suite, OWASP ZAP, and Acunetix scan the live application for weaknesses that could be exploited. Dynamic analysis is useful for identifying vulnerabilities that may not be apparent from the source code alone.

Example: Using OWASP ZAP for Dynamic Analysis

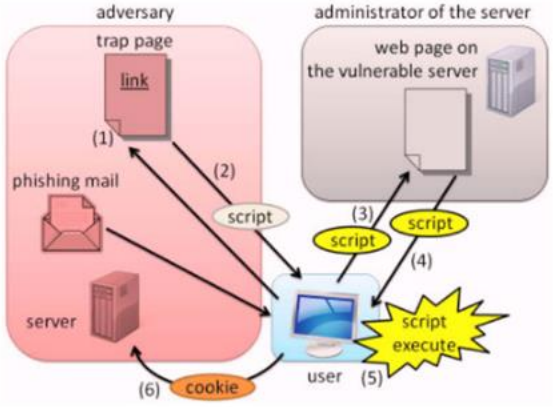
```
# Start the OWASP ZAP tool
zap.sh

# Perform a spider scan to discover the application's structure
zap-cli spider http://example.com

# Perform an active scan to identify vulnerabilities
zap-cli active-scan http://example.com
```

In this example, OWASP ZAP is used to perform both a spider scan and an active scan on a web application to identify potential vulnerabilities.

5. Classification of Input Validation Vulnerabilities

We propose a new classification system for input validation vulnerabilities based on the context in which the data is used:

1. **SQL Injection:** Occurs when user input is directly included in SQL queries without proper validation. This can lead to unauthorized access to the database.
2. **Cross-Site Scripting (XSS):** Occurs when user input is included in web pages without proper sanitization, allowing attackers to execute malicious scripts in the user's browser.
3. **XPath Injection:** Similar to SQL injection, but targets XML databases, allowing attackers to manipulate or retrieve sensitive data.
4. **LDAP Injection:** Occurs when unvalidated user input is included in LDAP queries, potentially exposing or manipulating directory services.
5. **NoSQL Injection:** Targets NoSQL databases, exploiting unvalidated inputs to execute arbitrary database operations.



6. **Header Injection:** Occurs when unvalidated user inputs are included in HTTP headers, potentially leading to session hijacking or other attacks.
7. **Email Injection:** Involves injecting malicious content into email headers, potentially leading to email spoofing or other exploits.
8. **Path Traversal:** Occurs when unvalidated inputs are used to access file system paths, potentially exposing or manipulating files.

6. Evaluation Metrics

To evaluate the effectiveness of various techniques in detecting and preventing input validation vulnerabilities, we use common metrics such as precision, recall, and F-measure. These metrics help in assessing the accuracy and efficiency of the detection tools.

Example: Preventing SQL Injection

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "database";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$stmt = $conn->prepare("SELECT * FROM users WHERE username = ?");
$stmt->bind_param("s", $userInput);
$userInput = $_GET['userInput'];
$stmt->execute();
$result = $stmt->get_result();

if ($result->num_rows > 0) {
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " . $row["username"]. "<br>";
    }
} else {
    echo "0 results";
}
$stmt->close();
$conn->close();
?>
```

In this example, prepared statements are used to prevent SQL injection by separating the SQL logic from the user input.

7. Analysis and Discussion

Our analysis reveals that while static analysis tools are effective in early detection, they may produce false positives. Dynamic analysis tools, on the other hand, are better at identifying runtime vulnerabilities but can be limited by the scope of the testing environment. A combination of both techniques often provides the best results.

8. Limitations and Future Work

Current approaches to mitigating input validation vulnerabilities have limitations, including false positives in static analysis and the need for extensive testing environments for dynamic analysis. Future research should focus on improving the accuracy of detection tools and developing comprehensive frameworks that integrate both static and dynamic analysis techniques.

9 .Conclusion

Input validation is a critical aspect of web security, and addressing vulnerabilities related to it is essential for preventing injection attacks and data manipulation. By providing a systematic review and new classification of input validation vulnerabilities, this paper aims to enhance the understanding of these issues and promote the development of more effective detection and prevention techniques.

References

[1]. R. Fielding. (1999). Hypertext Transfer Protocol--HTTP/1.1. IETF RFC 2616. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[2]. D. Kopec, ‘‘History of web programming,’’ in Dart for Absolute Beginners. Cham, Switzerland: Springer, 2014, pp. 275–286

[3]. J. Fonseca, M. Vieira, and H. Madeira, ‘‘The web attacker perspective—A field study,’’ in Proc. IEEE 21st Int. Symp. Softw. Rel. Eng., Nov. 2010, pp. 299–308.

[4]. A. Delaitre, B. Stivalet, E. Fong, and V. Okun, ‘‘Evaluating bug finders–test and measurement of static code analyzers,’’ in Proc. IEEE/ACM 1st Int. Workshop Complex Faults Failures Large Softw. Syst. (COUFLESS), May 2015, pp. 14–20.

[5]. T. Scholte, D. Balzarotti, and E. Kirda, ‘‘Have things changed now? An empirical study on input validation vulnerabilities in web applications,’’ Comput. Secur., vol. 31, no. 3, pp. 344–356, May 2012.

[6]. D. Wichers. Leadership of the OWASP Top 10 Project. Accessed: Jun. 3, 2021. [Online]. Available: https://owasp.org/www-project-top-ten/

[7]. Leadership of the OWASP Top 10 Project. Accessed: Jun. 3, 2021. [Online]. Available: https://github.com/owasp-top/owasp-top-2007

[8]. Leadership of the OWASP Top 10 Project. Accessed: Jun. 3, 2021. [Online]. Available: https://owasp.org/wwwpdfarchive/OWASP_AppSec_Research_2010_OWASP_Top_10_by_Wichers.pdf

[9]. Leadership of the OWASP Top 10 Project. Accessed: Jun. 3, 2021. [Online]. Available: https://owasp.org/www-pdf-archive/OWASP_Top_10_-_2013.pdf

[10]. Leadership of the OWASP Top 10 Project. Accessed: Jun. 3, 2021. [Online]. Available: https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf

[11]. O. B. Fredj, O. Cheikhrouhou, M. Krichen, H. Hamam, and A. Derhab, ‘‘An OWASP top ten driven survey on web application protection methods,’’ in Proc. Int. Conf. Risks Secur. Internet Syst. Cham, Switzerland: Springer, 2020, pp. 235–252.

[12]. G. K. Pannu, ‘‘A survey on web application attacks,’’ Int. J. Comput. Sci. Inf. Technol., vol. 5, no. 3, pp. 1–5, 2014.

[13]. H. Atashzar, A. Torkaman, M. Bahrololum, and M. H. Tadayon, ‘‘A survey on web application vulnerabilities and countermeasures,’’ in Proc. 6th Int. Conf. Comput. Sci. Converg. Inf. Technol. (ICCIT), 2011, pp. 647–652.

[14]. O. B. Al-Khurafi and M. A. Al-Ahmad, ‘‘Survey of web application vulnerability attacks,’’ in Proc. 4th Int. Conf. Adv. Comput. Sci. Appl. Technol. (ACSAT), Dec. 2015, pp. 154–158.
[15]. C. Meghana, B. Chaitra, and V. Nagaveni, ‘‘Survey on—Web application attack detection using data mining techniques,’’ J. Comput., Internet Netw. Secur., vol. 4, no. 2, pp. 154–158, 2018.
[16]. M. Khari, ‘‘Web-application attacks: A survey,’’ in Proc. 3rd Int. Conf. Comput. Sustain. Global Develop. (INDIACom), 2016, pp. 2187–2191.
[17]. N. ElBachirElMoussaid and A. Toumanari, ‘‘Web application attacks detection: A survey and classification,’’ Int. J. Comput. Appl., vol. 103, no. 12, pp. 1–6, Oct. 2014.
[18]. SANS. Private U.S. for-Profit Company. Accessed: Jun. 3, 2021. [Online]. Available: https://www.sans.org/top25-software-errors/
[19]. I. V. de Sousa Medeiros, ‘‘Detection of vulnerabilities and automatic protection for web applications,’’ Ph.D. dissertation, Dept. Comput. Sci., Universidade de Lisboa, Portugal, 2016.
[20]. D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, ‘‘Saner: Composing static and dynamic analysis to validate sanitization in web applications,’’ in Proc. IEEE Symp. Secur. Privacy (SP), May 2008, pp. 387–401.
[21]. A. Algaith, P. Nunes, F. Jose, I. Gashi, and M. Vieira, ‘‘Finding SQL injection and cross site scripting vulnerabilities with diverse static analysis tools,’’ in Proc. 14th Eur. Dependable Comput. Conf. (EDCC), Sep. 2018, pp. 57–64.
[22]. J. Dahse and T. Holz, ‘‘Simulation of built-in php features for precise static code analysis,’’ in Proc. NDSS Symp., vol. 14. Princeton, NJ, USA: Citeseer, 2014, pp. 23–26.
[23]. D. Hauzar and J. Kofron, ‘‘Framework for static analysis of php applications,’’ in Proc. 29th Eur. Conf. Object-Oriented Program. (ECOOP). Dagstuhl, Germany: Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015, pp. 689–711.
[24]. N. L. de Poel, F. B. Brokken, and G. R. R. de Lavalette, ‘‘Automated security review of PHP web applications with static code analysis,’’ M.S. thesis, Dept. Comput. Sci., vol. 5, 2010.
[25]. P. J. C. Nunes, J. Fonseca, and M. Vieira, ‘‘PhpSAFE: A security analysis tool for OOP web application plugins,’’ in Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw., Jun. 2015, pp. 299–306.
[26]. N. Munaiah, ‘‘Assisted discovery of software vulnerabilities,’’ in Proc. 40th Int. Conf. Softw. Eng., Companion, May 2018, pp. 464–467.
[27]. P. E. Black, P. E. Black, M. Kass, M. Koo, and E. Fong, ‘‘Source code security analysis tool functional specification version 1.0,’’ U.S. Dept. Commerce, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. 500-268, 2007.
[28]. C. Cao, N. Gao, P. Liu, and J. Xiang, ‘‘Towards analyzing the input validation vulnerabilities associated with Android system services,’’ in Proc. 31st Annu. Comput. Secur. Appl. Conf., Dec. 2015, pp. 361–370.
[29]. A. Z. Baset and T. Denning, ‘‘IDE plugins for detecting input-validation vulnerabilities,’’ in Proc. IEEE Secur. Privacy Workshops (SPW), May 2017, pp. 143–146.
[30]. G. Wassermann and Z. Su, ‘‘Static detection of cross-site scripting vulnerabilities,’’ in Proc. 13th Int. Conf. Softw. Eng. (ICSE), 2008, pp. 171–180.
[31]. T. Scholte, W. Robertson, D. Balzarotti, and E. Kirda, ‘‘An empirical analysis of input validation mechanisms in web applications and languages,’’ in Proc. 27th Annu. ACM Symp. Appl. Comput., Mar. 2012, pp. 1419–1426.
[32]. E. Ufuktepe and T. Tuglular, ‘‘Estimating software robustness in relation to input validation vulnerabilities using Bayesian networks,’’ Softw. Quality J., vol. 26, no. 2, pp. 455–489, Jun. 2018.
[33]. M. Alkhalaf, S. R. Choudhary, M. Fazzini, T. Bultan, A. Orso, and C. Kruegel, ‘‘ViewPoints: Differential string analysis for discovering client- and server-side input validation inconsistencies,’’ in Proc. 18th Int. Conf. Fundam. Approaches Softw. Eng. (FASE), London, U.K., Apr. 2015, pp. 56–71
[34]. A. F. Tappenden and J. Miller, ‘‘Automated cookie collection testing,’’ in Proc. 6th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng., Sep. 2007, pp. 190–199.