WebGoat Dynamic Application Security Testing Report

Daniel Cordeiro Marques

University of San Diego

CSOL-560-01-FA21 - Secure Software Design Development

Professor Ashton Mozano

October 25, 2021

Table of Contents

WebGoat Dynamic Application Security Testing Report

This report describes the effort to perform a Dynamic Application Security Testing (DAST) on WebGoat using OWASP ZAP. The work outlines the methodology used to perform the test, the vulnerabilities identified during the exercise, and recommendations for remediation.

Overview

WebGoat Overview

According to *OWASP WebGoat* (n.d.), "WebGoat is a deliberately insecure application that allows interested developers just like you to test vulnerabilities commonly found in Java-based applications that use common and popular open-source components." As such, WebGoat is a viable tool to baseline security tools and understand their features and limitations. This Dynamic Application Security Testing (DAST) covers WebGoat version 8.2.2.

Methodology

This DAST consisted of an automated scan using OWASP ZAP, a tool that checks for multiple vulnerabilities in web applications based on a set of plugins and policies. To perform this DAST, the tester manually navigated through the application to identify its pages and entry points to complete this DAST, expanding the mapping using the automated spider function. Next, the "full scan" policy was executed against the application.

Severity ratings. Once the scan was completed, the tester used the Common Vulnerability Scoring System (CVSS) version 3.1 base score to rate the identified vulnerabilities. Table 1 demonstrates the severity according to FIRST (n.d.).

Table 1.

Vulnerability Classification Ratings based on FIRST (n.d.).

Severity	CVSS Score
Critical	9.0 - 10.0
High	7.0 - 8.9
Medium	4.0 - 6.9
Low	0.1 - 3.9
Informational (None)	0.0

Summary Of Findings

Most of the findings identified during this DAST with the highest severity are associated with input validation vulnerabilities. Table 2 outlines the vulnerabilities observed during this exercise.

Table 2.

Vulnerabilities Observed.

Finding	Severity	Vulnerable Instances (Total)
Finding H1. Reflected Cross-Site Scripting (XSS)	**High**	1
Finding H2. SQL Injection	**High**	13
Finding M1. Format String Error	**Medium**	6
Finding M2. Vulnerable JavaScript Library In Use	**Medium**	5
Finding L1. Cookie Without “HttpOnly” Flag	**Low**	3
Finding L2. Cookie Without “SameSite” Attribute	**Low**	3
Finding I1. Sensitive Information Disclosure In the URL	**Informational**	2

Analysis of Scan Results

This section describes the results of the scans performed by the tester using OWASP ZAP. Each finding is represented by a finding ID and a title, its severity rating, and CWE ID, a description that outlines the impact and potential attack vectors, and a list of affected instances with the associated URL and additional relevant information.

The proposed remediation steps consider standard practices for Java-based applications; however, the developers must exercise caution and consider the impact of these changes on application performance and integration with other systems.

Finding H1. Reflected Cross-Site Scripting (XSS)

Severity. High.

CWE ID. 79 (*CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-Site Scripting')*, 2006).

Description. Reflected Cross-Site Scripting attacks (also known as Type-II or non-persistent XSS) are when the application reflects the vulnerable parameter to the browser without properly validating input data. These values are not stored server-side.

Search results that include queried values in the URL and error messages that return the user's input are typical examples of potentially vulnerable instances. For example, attackers may use reflected XSS combined with other attacks (such as social engineering) to email a crafted URL that triggers an error message and executes malicious code when rendered by the user's browser.

Potential impact can vary from modifying web page contents to obtaining the user's session information and installing malware in the system, as Abrams (2021) reported.

Affected Instances. Table 3 lists the URLs and parameters affected by this vulnerability.

Table 3.

URLs And Parameters affected by Finding H1.

URL	Request Type	Parameter	Attack
http://192.168.1.192:8080/WebGoat/CrossSiteScripting/attack5a?QTY1=1&QTY2=1&QTY3=1&QTY4=1&field1=%3Cimg+src%3Dx+onerror%3Dalert%281%29%3B%3E&field2=111	GET	field1	<img src=x onerror=alert(1);>

Remediation. The application should not trust user-provided data, and therefore should not insert data in the HTML code without sanitization. *Cross Site Scripting Prevention Cheat Sheet* (n.d.) recommends that developers use data encoding with care and consider the context and syntax applicable to "the part of the HTML document you're putting untrusted data into." For instance, developers should use different encoding schemes for data the browser will render as the contents of an HTML element versus HTML attributes.

Finding H2. SQL Injection

Severity. High.

CWE ID. 89 (*CWE-89: Improper Neutralization of Special Elements Used in an SQL Command ('SQL Injection')*, 2006).

Description. SQL injections occur when the application uses untrusted input data without proper validation to build SQL instructions. Users can leverage the affected parameters

to send commands to the database management system (DBMS), and execute instructions directly against the application's database.

The impact may vary according to the access granted to the application; as Kingthorin (n.d.) noted, "a successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system."

Affected Instances. Table 4 lists the URLs and parameters affected by this vulnerability.

Table 4.

URLs And Parameters affected by Finding H2.

URL	Request Type	Parameter	Attack
http://192.168.1.192:8080/WebGoat/SqlInjection/assignment5a	POST	injection	'1' = '1' AND '1'='1' --
http://192.168.1.192:8080/WebGoat/SqlInjection/attack10	POST	action_string	fghjm OR 1=1 --
http://192.168.1.192:8080/WebGoat/SqlInjection/attack8	POST	auth_tan	69069134/2
http://192.168.1.192:8080/WebGoat/SqlInjectionAdvanced/attack6a	POST	userid_6a	' OR '1'='1' --
http://192.168.1.192:8080/WebGoat/SqlInjectionAdvanced/challenge_Login	POST	password_login	AND 1=1 --
http://192.168.1.192:8080/WebGoat/SqlInjectionMitigations/servers?column=%27	GET	column	'

http://192.168.1.192:8080/WebGoat/SqlInjection/assignment5a	POST	account	"
http://192.168.1.192:8080/WebGoat/SqlInjection/assignment5a	POST	injection	;
http://192.168.1.192:8080/WebGoat/SqlInjection/assignment5a	POST	operator	'
http://192.168.1.192:8080/WebGoat/SqlInjection/assignment5b	POST	userid	;
http://192.168.1.192:8080/WebGoat/SqlInjection/attack3	POST	query	;
http://192.168.1.192:8080/WebGoat/SqlInjection/attack	POST	name	'
http://192.168.1.192:8080/WebGoat/SqlInjection/attack9	POST	name	'

Remediation. Developers should follow a layered approach to defend the application against SQL injection attacks. The first line of defense is to use prepared statements with parameterized queries; this measure forces developers to define SQL queries and parameters separately, allowing "the database to distinguish between code and data, regardless of what user input is supplied" and preventing attackers from changing "the intent of a query, even if SQL commands are inserted by an attacker" (*SQL Injection Prevention Cheat Sheet*, n.d.).

If prepared statements are not a viable option (potentially due to performance concerns), developers may use stored procedures that do not dynamically generate unsafe SQL code. Additionally, OWASP's *SQL Injection Prevention Cheat Sheet* (n.d.) recommends that developers can consider the use of allow-list for input validation and escaping untrusted input.

For the next defense layer, developers should apply the principle of least privilege. First, the application should access the database with only the minimum privilege required to perform its activities, restricted exclusively to the necessary databases. Using views can also limit data exposure in case an attacker successfully exploits a SQL injection.

Finding M1. Format String Error

Severity. Medium.

CWE ID. 134 (*CWE-134: Use of Externally-Controlled Format String*, 2006).

Description. As defined by Meir555 (n.d.), "the Format String exploit occurs when the submitted data of an input string is evaluated as a command by the application." Format string vulnerabilities are typically associated with C/C++, but applications written in Java or PHP are also exposed to these issues. These vulnerabilities occur when the application does not properly validate the untrusted input before passing to a format function, which in turn parses the conversion characters.

Threats that can successfully exploit this vulnerability may gain control of the application flow, potentially allowing the attacker to execute arbitrary code in the server, read values from the stack, and cause a denial of service condition.

Affected Instances. Table 5 lists the URLs and parameters affected by this vulnerability.

Table 5.

URLs And Parameters affected by Finding M1.

URL	Request Type	Parameter	Attack
http://192.168.1.192:8080/WebGoat/CrossSiteScripting/attack1	POST	answer_xss_1	ZAP%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s
http://192.168.1.192:8080/WebGoat/CrossSiteScripting/attack6a	POST	DOMTestRoute	ZAP%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s
http://192.168.1.192:8080/WebGoat/IDOR/login	POST	username	ZAP%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s
http://192.168.1.192:8080/WebGoat/JWT/quiz	POST	question_0_solution	ZAP%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s
http://192.168.1.192:8080/WebGoat/SqlInjectionAdvanced/attack6b	POST	userid_6b	ZAP%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s%n%s
http://192.168.1.192:8080/WebGoat/xxe	POST	text	ZAP%n%s%n%s%

/blind			n%s%n%s%n%s% n%s%n%s%n%s% n%s%n%s%n%s% n%s%n%s%n%s% n%s%n%s%n%s% n%s%n%s%n%s

Remediation. Format string errors can be addressed by improving the application's input validation capabilities, which takes into account the data's syntax and semantics of the data. Developers should consider the use of server-side allow-lists; as noted in *Input Validation Cheat Sheet* (n.d.), "allow list validation involves defining exactly what IS authorized, and by definition, everything else is not authorized", and is effective when handling structured data.

Additionally, developers should address error handling considerations to avoid exposing application information unnecessarily. For applications using SpringMVC, the *Error Handling Cheat Sheet* (n.d.) proposes a global error handler that extends the class "java.lang.Exception" using the annotation "ExceptionHandler".

Finding M2. Vulnerable JavaScript Library In Use

Severity. Medium.

CWE ID. 829 (*CWE-829: Inclusion of Functionality From Untrusted Control Sphere*, 2010).

Description. The scanner identified that WebGoat uses outdated versions of JavaScript libraries. These libraries are known to be affected by multiple vulnerabilities, some dating as far back as 2016, that may lead to including cross-site scripting and code execution attacks.

Using components with known vulnerabilities may expose users to vulnerabilities with reliable exploits; however, the impact depends on the affected component.

Affected Instances. Table 6 lists the URLs and parameters affected by this vulnerability.

Table 6.

Libraries affected by Finding M2.

URL	Library	Version	CVEs
http://192.168.1.192:8080/WebGoat/js/libs/bootstrap.min.js	Bootstrap	v3.1.1	CVE-2019-8331 CVE-2018-14041 CVE-2018-14040 CVE-2018-14042
http://192.168.1.192:8080/WebGoat/js/libs/jquery-2.1.4.min.js	JQuery (minified)	2.1.4	CVE-2020-11023 CVE-2020-11022 CVE-2015-9251 CVE-2019-11358
http://192.168.1.192:8080/WebGoat/js/libs/jquery-ui-1.10.4.js	jQuery UI	1.10.3	CVE-2016-7103
http://192.168.1.192:8080/WebGoat/js/libs/jquery.min.js	JQuery (minified)	3.4.1	CVE-2020-11023 CVE-2020-11022
http://192.168.1.192:8080/WebGoat/lesson_js/bootstrap.min.js	Bootstrap	v3.1.1	CVE-2019-8331 CVE-2018-14041 CVE-2018-14040 CVE-2018-14042

Remediation. Keeping track of the dependencies' security state is vital to maintain the application's security posture; therefore, the *Vulnerable Dependency Management Cheat Sheet* (n.d.) recommends that developers "perform automated analysis of the dependencies from the birth of the project." This "shifting left" approach to security helps reduce the overall cost of

addressing potential issues with libraries by acting early in the development process (*DevOps Tech: Shifting Left on Security*, 2021).

The management process must consider the impact of the library on the application's functional requirements, as well as mechanisms that might mitigate any associated risks. For instance, a security researcher might publish a critical vulnerability in a third-party library used by the application to deliver its primary services; if the vendor chooses to not address the vulnerability, the application owners may require the developers to review the application and implement countermeasures to address potential risks associated with the use of the said library. As part of the process, developers must trace application calls to the affected library, and ensure the application takes appropriate steps to prevent attackers from triggering the vulnerability or minimizing the impact in case of success.

Finding L1. Cookie Without "HttpOnly" Flag

Severity. Low.

CWE ID. 1004 (*CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag*, 2017).

Description. The scanner identified cookies carrying sensitive information, such as SessionID cookies, without the HttpOnly flag. As "*CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag"* (2017) notes, this flag prevents client-side scripts from reading the contents of the cookie, providing an additional layer of defense against cross-site scripting attacks.

Affected Instances. Table 7 lists the URLs and parameters affected by this vulnerability.

Table 7.

URLs And Cookies affected by Finding L1.

URL	Request Type	Cookie
http://192.168.1.192:8080/WebGoat/	GET	JSESSIONID
http://192.168.1.192:8080/WebGoat/JWT/votings/login?user=Guest	GET	access_token
http://192.168.1.192:8080/WebGoat/login	POST	JESSIONID

Remediation. The application should set the HttpOnly flag on all cookies handling sensitive data. In Java, developers can achieve that by using the methods “setHttpOnly” and “isHttpOnly”, available in the Cookie interface (for instance, “cookie.setHttpOnly(true);”).

Finding L2. Cookie Without “SameSite” Attribute

Severity. Low.

CWE ID. 1275 (*CWE-1275: Sensitive Cookie With Improper SameSite Attribute*, 2020).

Description. The scanner identified cookies carrying sensitive information, such as SessionID cookies, without the SameSite attribute. This attribute determines how the browser transmits these cookies to another domain as part of a cross-domain request.

Leveraging the SameSite attribute serves as an additional layer of defense against cross-site request forgery (CSRF) attacks, as it can prevent the browser from adding sensitive cookies to cross-site POST requests.

Affected Instances. Table 8 lists the URLs and parameters affected by this vulnerability.

Table 9.

URLs And Cookies affected by Finding L2.

URL	Request Type	Cookie
http://192.168.1.192:8080/WebGoat/	GET	JSESSIONID
http://192.168.1.192:8080/WebGoat/JWT/votings/login?user=Guest	GET	access_token
http://192.168.1.192:8080/WebGoat/login	POST	JESSIONID

Remediation. The application should set the SameSite attribute to “Lax” or “Strict” on all cookies handling sensitive data. Set the SameSite attribute of a sensitive cookie to 'Lax' or 'Strict'. This is classified as highly effective by *CWE-1275: Sensitive Cookie With Improper SameSite Attribute* (2020), as it “ instructs the browser to apply this cookie only to same-domain requests”.

Setting the SameSite attribute to “Lax” still allows the browser to send cookies to top-level domains using HTTP methods that do not cause a change of state, such as GET, HEAD, OPTIONS, and TRACE; whereas using “Strict” will not allow the browser to send cookies to third-parties.

Finding I1. Sensitive Information Disclosure In the URL

Severity. Informational.

CWE ID. 200 (*CWE-200: Exposure of Sensitive Information to an Unauthorized Actor*, 2006).

Description. By sending sensitive information in the URL the application exposes data to unwanted third parties even when using encrypted communications. Information transmitted in

the URL can be found in many locations after the request is completed, including the referrer header, application logs, browser history, and browser cache.

For instance, if the application exposes the user's session ID, threats might use this vulnerability to perform forced browsing attacks and impersonate the user, gaining unauthorized access to the application.

Affected Instances. Table 10 lists the URLs and parameters affected by this vulnerability.

Table 10.

URLs And Parameters affected by Finding H1.

URL	Request Type	Parameter
http://192.168.1.192:8080/WebGoat/JWT/final/delete?token=eyJ0eXAiOiJKV1QiLCJraWQiOiJ3ZWJnb2F0X2tleSIsImFsZyI6IkhTMjU2In0.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEJ1aWxkZXIiLCJpYXQiOjE1MjQyMTA5MDQsImV4cCI6MTYxODkwNTMwNCwiYXVkIjoid2ViZ29hdC5vcmciLCJzdWIiOiJqZXJyeUB3ZWJnb2F0LmNvbSIsInVzZXJuYW1lIjoiSmVycnkiLCJFbWFpbCI6ImplcnJ5QHdlYmdvYXQuY29tIiwiUm9sZSI6WyJDYXQiXX0.CgZ27DzgVW8gzc0n6izOU638uUCi6UhiOJKYzoEZGE8	GET	token
http://192.168.1.192:8080/WebGoat/JWT/final/delete?token=eyJ0eXAiOiJKV1QiLCJraWQiOiJ3ZWJnb2F0X2tleSIsImFsZyI6IkhTMjU2In0.eyJpc3MiOiJXZWJHb2F0IFRva2VuIEJ1aWxkZXIiLCJpYXQiOjE1MjQyMTA5MDQsImV4cCI6MTYxODkwNTMwNCwiYXVkIjoid2ViZ29hdC5vcmciLCJzdWIiOiJqZXJyeUB3ZWJnb2F0LmNvbSIsInVzZXJuYW1lIjoiSmVycnkiLCJFbWFpbCI6ImplcnJ5QHdlYmdvYXQuY29tIiwiUm9sZSI6WyJDYXQiXX0.CgZ27DzgVW8gzc0n6izOU638uUCi6UhiOJKYzoEZGE8	POST	token

Remediation. If possible, remove sensitive information from URLs. Additionally, disable caching for responses containing sensitive information.

Mitigations and Recommendations

In addition to the specific recommendations to remediate the previously outlined vulnerabilities, the developers should improve their input validation strategy. The findings identified by this DAST with the highest severity are directly associated with the application's ability to sanitize data before processing. The development team can benefit from using data type validators already offered by Java and SpringMVC and trusted third-party security libraries.

Finally, establishing an automated process to check for outdated and vulnerable dependencies (such as GitHub's Dependabot) can help developers maintain the application's security posture while reducing the cost of addressing potential vulnerabilities inherited from these libraries.

References

Abrams, L. (2021, August 23). *Phishing campaign uses UPS.com XSS vuln to distribute malware*. Bleeping Computer. Retrieved October 20, 2020, from https://www.bleepingcomputer.com/news/security/phishing-campaign-uses-upscom-xss-vuln-to-distribute-malware/

Cross Site Scripting Prevention Cheat Sheet. (n.d.). OWASP Cheat Sheet Series. Retrieved October 20, 2021, from https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'). (2006). Common Weaknesses Database. https://cwe.mitre.org/data/definitions/79.html

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'). (2006). Common Weaknesses Database. https://cwe.mitre.org/data/definitions/89.html

CWE-134: Use of Externally-Controlled Format String. (2006, 07 19). Common Weakness Enumeration. Retrieved October 25, 2021, from https://cwe.mitre.org/data/definitions/134.html

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor. (2006). Common Weakness Database. Retrieved October 25, 2021, from https://cwe.mitre.org/data/definitions/200.html

CWE-829: Inclusion of Functionality from Untrusted Control Sphere. (2010). Common Weakness Enumeration. Retrieved October 25, 2021, from https://cwe.mitre.org/data/definitions/829.html

CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag. (2017). Common Weakness Enumeration. Retrieved October 25, 2021, from https://cwe.mitre.org/data/definitions/1004.html

CWE-1275: Sensitive Cookie with Improper SameSite Attribute. (2020). Common Weakness Enumeration. Retrieved October 25, 2021, from https://cwe.mitre.org/data/definitions/1275.html

DevOps tech: Shifting left on security. (2021, October 14). Google Cloud Architecture Center. Retrieved October 25, 2021, from https://cloud.google.com/architecture/devops/devops-tech-shifting-left-on-security#how_to_implement_improved_security_quality

Error Handling Cheat Sheet. (n.d.). OWASP Cheat Sheet Series. Retrieved October 25, 2021, from https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html

FIRST. (n.d.). *Common Vulnerability Scoring System version 3.1 Specification Document*. Common Vulnerability Scoring System (CVSS-SIG). Retrieved October 23, 2021, from https://www.first.org/cvss/v3.1/specification-document

Input Validation Cheat Sheet. (n.d.). OWASP Cheat Sheet Series. Retrieved October 25, 2021, from https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

kingthorin. (n.d.). *SQL Injection*. OWASP - Attacks. Retrieved October 23, 2021, from https://owasp.org/www-community/attacks/SQL_Injection

meir555. (n.d.). *Format string attack*. OWASP Attacks. Retrieved October 25, 2021, from https://owasp.org/www-community/attacks/Format_string_attack

OWASP WebGoat. (n.d.). The OWASP Foundation. Retrieved October 25, 2021, from https://owasp.org/www-project-webgoat/

SQL Injection Prevention Cheat Sheet. (n.d.). OWASP Cheat Sheet Series. Retrieved October 24, 2021, from https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Vulnerable Dependency Management Cheat Sheet. (n.d.). OWASP Cheat Sheet Series. Retrieved October 25, 2021, from https://cheatsheetseries.owasp.org/cheatsheets/Vulnerable_Dependency_Management_Cheat_Sheet.html