**VULNERABILITY REPORT FOR OWASP WEBGOAT PROJECT**

**DATE**: 05-09-2024

**PRESENTED BY**: Kota Hima Manogna (22B01A0597)

**Department**: Computer Science and Engineering (section-B)

**Email**: [22b01a0597@svecw.edu.in](mailto:22b01a0597@svecw.edu.in)

(personal): [manognakota2005@gmail.com](mailto:manognakota2005@gmail.com)

INTRODUCTION:

**Project Name**: OWASP WebGoat

**Purpose**: The purpose of this report is to identify and categorize vulnerabilities found in the OWASP WebGoat project using Semgrep, and provide recommendations for mitigating true positives.

**Summary Of Findings:**

These are the total of true positives which I have identified in the OWASP WebGoat.

|  |  |
| --- | --- |
| SEVERITY | FINDINGS |
| High | 67 |
| Medium | 169 |
| Low | 0 |
| TOTAL | 236 |

|  |  |
| --- | --- |
| CONFIDENCE | FINDINGS |
| High | 35 |
| Medium | 132 |
| Low | 69 |
| TOTAL | 236 |

**Detailed Summary:**

High Severity Vulnerabilities Summary Table

| **Vulnerability Type** | **Rules** | **Count** |
| --- | --- | --- |
| **High Severity** |  | 67 |
| SQL Injection | formatted-sql-string,  java-sql-sqli,  tainted-sql-string | 12 |
| Path Traversal | httpservlet-path-traversal,  spring-tainted-path-traversal | 9 |
| Insecure Deserialization | objectinputstream-deserialization-spring | 14 |
| Cross-Site Request Forgery (CSRF) | insecure-document-method | 2 |
| Server-Side Request Forgery (SSRF) | tainted-ssrf-spring-add | 10 |
| Cross-Site Scripting (XSS) | formatted-sql-string | 1 |
| Improper Validation | tainted-url-host | 2 |
| Remote Code Execution (RCE) | deepsemgrep | 1 |

Here is the list of vulnerabilities I have found based on the severity, the rules for the vulnerability and their mitigations.

High Severity Vulnerabilities:

1. **SQL Injection : 12 Findings**
   * **Description**: SQL Injection occurs when an attacker can execute arbitrary SQL queries on the database through unsanitized input. This can lead to unauthorized access, data leakage, or data manipulation.
   * **Mitigation**: Use parameterized queries and prepared statements to prevent unsanitized data from entering SQL queries.
   * **Rules**:
     + formatted-sql-string
     + tainted-sql-string
     + java-sql-sqli
2. **Path Traversal : 9 Findings**
   * **Description**: Path Traversal allows attackers to access files and directories stored outside the intended directory by manipulating the file path. This can expose sensitive files or allow unauthorized access.
   * **Mitigation**: Sanitize and validate file paths to ensure they point to authorized directories only.
   * **Rules**:
     + httpservlet-path-traversal
     + tainted-file-path
3. **Remote Code Execution (RCE) : 1 Finding**
   * **Description**: Remote Code Execution enables attackers to run arbitrary code on the server or client, often leading to complete control over the affected system.
   * **Mitigation**: Use strong data validation and limit the classes allowed during deserialization to trusted sources only.
   * **Rules**:
     + insecure-deserialization
     + objectinputstream-deserialization-spring
4. **Insecure Deserialization: 15 Findings**
   * **Description**: Insecure Deserialization allows attackers to manipulate serialized data to execute arbitrary code or alter application behavior, often leading to RCE or data corruption.
   * **Mitigation**: Implement strict input validation and use safe deserialization libraries that enforce object integrity.
   * **Rules**:
     + objectinputstream-deserialization-spring
     + spring-tainted-path-traversal
     + deepsemgrep
5. **Cross-Site Scripting (XSS) : 1 Finding**
   * **Description**: Cross-Site Scripting involves injecting malicious scripts into web pages viewed by other users, potentially leading to data theft, session hijacking, or other malicious activities.
   * **Mitigation**: Use output encoding for user input and implement Content Security Policy (CSP) to prevent the execution of injected scripts.
   * **Rules**:
     + formatted-sql-string
     + tainted-url-host
6. **Cross-Site Request Forgery (CSRF) : 2 Findings**
   * **Description**: CSRF tricks users into performing unintended actions on a web application where they are authenticated, potentially compromising user data or application security.
   * **Mitigation**: Use anti-CSRF tokens in forms and validate requests to ensure they originate from trusted sources.
   * **Rules**:
     + django-no-csrf-token
7. **Server-Side Request Forgery (SSRF) : 10 Findings**
   * **Description**: SSRF allows attackers to make requests from the server to internal resources or services, potentially exposing sensitive data or affecting internal services.
   * **Mitigation**: Restrict server requests to trusted domains and sanitize user-provided URLs before processing them.
   * **Rules**:
     + tainted-url-host
     + spring-tainted-path-traversal

8**. Improper Validation: 2 Findings**

* **Description**: Improper validation occurs when a system does not properly check the input provided by users, potentially leading to a range of vulnerabilities such as unauthorized access, data corruption, or other security issues. This often involves insufficient checks on data formats or values.
* **Mitigation**: Implement strict input validation and enforce proper data types and formats for all user inputs.
* **Rules**:
  + tainted-url-host
  + formatted-sql-string

Medium Severity Vulnerabilities:

Medium Severity Vulnerabilities Summary Table

| **Vulnerability Type** | **Rules** | **Count** |
| --- | --- | --- |
| **Medium Severity** |  | **167** |
| **Missing Security Controls** | missing-integrity, plaintext-http-link, cookie-missing-httponly, cookie-missing-secure-flag, cookie-missing-samesite, cookie-issecure-false | 25 |
| **Insecure Cryptography** | use-of-md5, weak-random | 7 |
| **Insecure Debugging Practices** | printstacktrace, active-debug-code | 8 |
| **SQL Injection** | jdbc-sqli, spring-sqli | 8 |
| **Insecure Deserialization** | object-deserialization | 2 |
| **Cross-Site Scripting (XSS)** | var-in-href, tainted-html-string-responsebody | 15 |
| **Prototype Pollution** | prototype-pollution-loop | 2 |
| **Missing CSRF Protection** | django-no-csrf-token | 75 |
| **Unrestricted Request Mapping** | unrestricted-request-mapping | 15 |
| **Tainted Data Usage in Session** | tainted-session-from-http-request | 1 |
| **Regular Expression Vulnerabilities** | detect-non-literal-regexp | 8 |

There are many vulnerabilities in the medium severity, in them the ones with the higher confidence were:

1. **Insecure Cryptography**: **7 Findings**

* **Description**: Insecure cryptography refers to the use of weak or outdated cryptographic algorithms, such as MD5 or insecure random number generators, which can lead to vulnerabilities such as data breaches or tampering.
* **Mitigation**: Use modern and strong cryptographic algorithms (e.g., SHA-256, AES-256) and secure random number generation to ensure data integrity and security.
* **Rules**:
  + - use-of-md5
    - weak-random

1. **Unrestricted Request Mapping**: **15 Findings**

* **Description**: Unrestricted request mapping allows unauthorized users to access sensitive endpoints, potentially exposing sensitive data or critical functionality.
* **Mitigation**: Implement proper access control mechanisms, such as role-based access control (RBAC), to restrict access to sensitive endpoints.
* **Rules**:
  + - unrestricted-request-mapping

1. **Prototype Pollution**: **2 Findings**

* **Description**: Prototype pollution can allow attackers to modify the base object prototype, affecting all objects of that type in the application and leading to potential code execution.
* **Mitigation**: Sanitize input and ensure that user data cannot alter object prototypes, particularly in JavaScript environments.
* **Rules**:
  + - prototype-pollution-loop

1. **Tainted HTML String in Response Body**: **1 Findings**
   * **Description**: Tainted HTML strings in response bodies can lead to Cross-Site Scripting (XSS) attacks if not properly sanitized or escaped.
   * **Mitigation**: Escape or sanitize all HTML content before rendering it in response bodies to prevent XSS attacks.
   * **Rules**:
     + tainted-html-string-response-body
2. **Cross-Site Request Forgery (CSRF)**: **75 Findings**

* **Description**: CSRF tricks users into performing unintended actions on a web application where they are authenticated, potentially compromising user data or application security.
* **Mitigation**: Use anti-CSRF tokens in forms and validate requests to ensure they originate from trusted sources.
* **Rules**:
  + - django-no-csrf-token

Low Severity Vulnerabilities:

There were no Low severity vulnerabilities.