Introduction to Web Application Security

Niman Ransindu

RED USERS

Introduction

This report details the findings from a security analysis conducted on WebGoat, an intentionally vulnerable web application designed to teach web application security. The analysis was performed using OWASP ZAP to identify common security vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).

Objective

The primary objective of this task was to deepen the understanding of web application vulnerabilities through hands-on analysis and exploitation of a deliberately vulnerable application, thereby learning effective strategies to mitigate such vulnerabilities.

Tools Used

- **WebGoat:** A deliberately insecure web application providing a realistic teaching and learning environment.
- **OWASP ZAP (Zed Attack Proxy):** An open-source security tool used for detecting vulnerabilities in web applications.

Setup the Web Application

Web Application: WebGoat

1. Run using Docker

Already have a browser and ZAP and/or Burp installed on your machine in this case you can run the WebGoat image directly using Docker.

Every release is also published on DockerHub.

```
docker run -it -p 127.0.0.1:8080:8080 -p 127.0.0.1:9090:9090 webgoat/webgoat
```

If you want to reuse the container, give it a name:

```
docker run --name webgoat -it -p 127.0.0.1:8080:8080 -p 127.0.0.1:9090:9090 webgoat/webgoat
```

As long as you don’t remove the container you can use:

```
docker start webgoat
```

2. Run using Docker with complete Linux Desktop

Instead of installing tools locally we have a complete Docker image based on running a desktop in your browser. This way you only have to run a Docker image which will give you the best user experience.

```
docker run -p 127.0.0.1:3000:3000 webgoat/webgoat-desktop
```

3. Standalone

Download the latest WebGoat release from https://github.com/WebGoat/WebGoat/releases

```
java -Dfile.encoding=UTF-8 -Dwebgoat.port=8080 -Dwebwolf.port=9090 -jar webgoat-2023.5.jar
```

Downloading and running WebGoat as a standalone Java application. Here's a step-by-step guide:

Step 1: Install Java

WebGoat requires Java to run. Follow these steps to install Java if it's not already installed.

1. **Update your package list**:

   ```
   sudo apt update
   ```

2. **Install OpenJDK (Java Development Kit)**: Install the necessary JDK (Java 11 is sufficient):

   ```
   sudo apt install openjdk-11-jdk
   ```

3. **Verify Java installation**: After installation, check the version to confirm:

   ```
   java -version
   ```

Step 2: Download WebGoat

Now, download the latest release of WebGoat.

1. **Go to the WebGoat releases page** on GitHub: WebGoat Releases
2. **Download the latest .jar file** (e.g., webgoat-server-<version>.jar).

Alternatively, you can use wget to download it directly:

```
wget https://github.com/WebGoat/WebGoat/releases/download/v8.2.0/webgoat-server-8.2.0.jar
```

Step 3: Run WebGoat

Once the .jar file is downloaded, you can run WebGoat directly using Java:

1. Navigate to the directory where the .jar file is located.
2. Run WebGoat with the following command:

   java -jar webgoat-server-<version>.jar

For example:

```
java -jar webgoat-server-8.2.0.jar
```

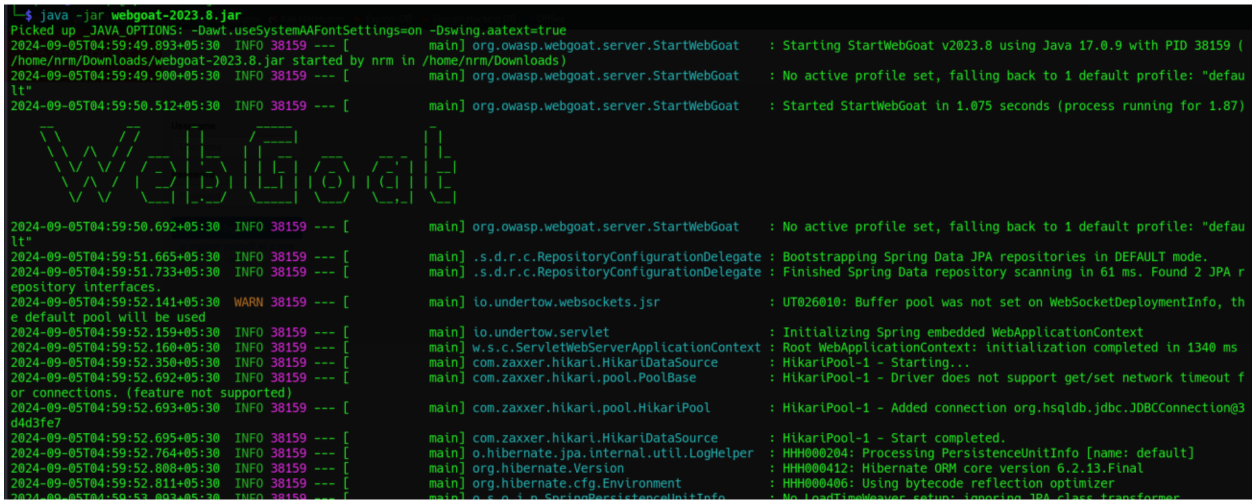
By default, WebGoat will run on port 8080.

Step 4: Access WebGoat

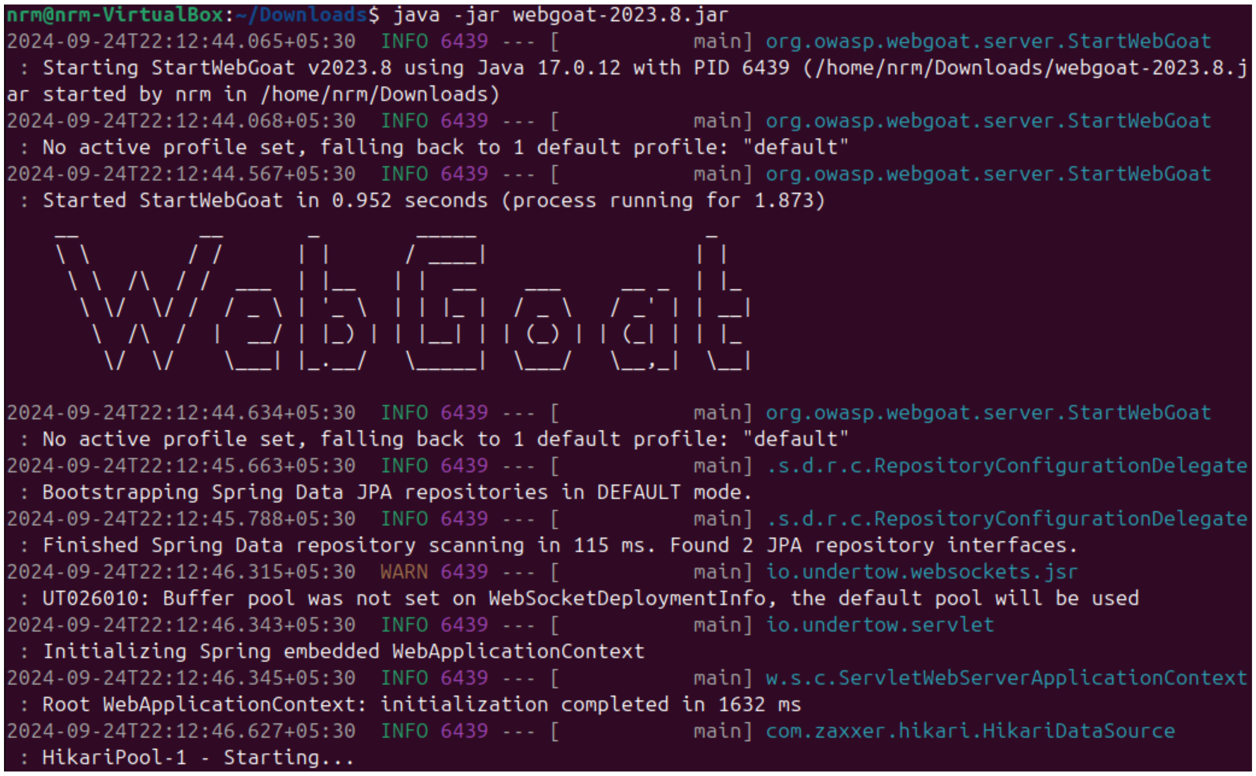
Once WebGoat is running, open a web browser in your Kali Linux VM and navigate to:

http://localhost:8080/WebGoat

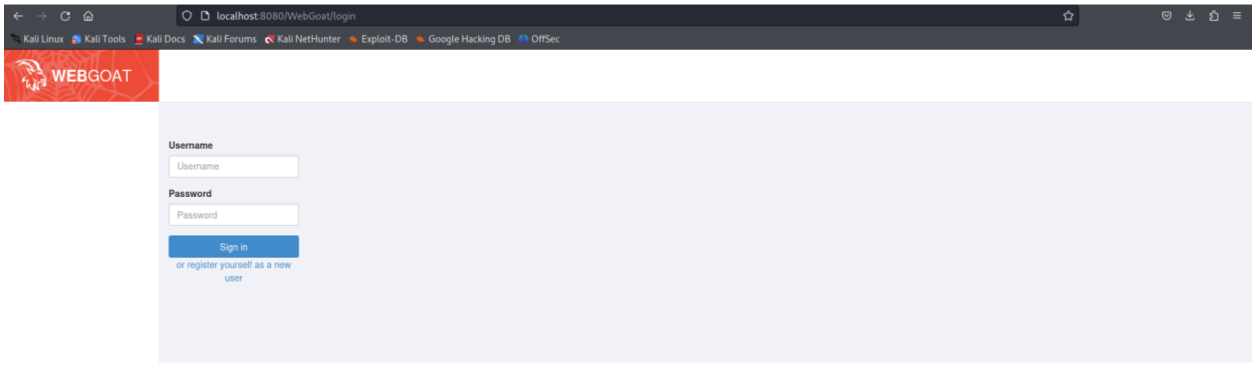
You should now have WebGoat up and running.



```
└─$ java -jar webgoat-2023.8.jar
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
2024-09-05T04:59:49.893+05:30  INFO 38159 --- [           main] org.owasp.webgoat.server.StartWebGoat    : Starting StartWebGoat v2023.8 using Java 17.0.9 with PID 38159 (
/home/nrm/Downloads/webgoat-2023.8.jar started by nrm in /home/nrm/Downloads)
2024-09-05T04:59:49.900+05:30  INFO 38159 --- [           main] org.owasp.webgoat.server.StartWebGoat    : No active profile set, falling back to 1 default profile: "defau
lt"
2024-09-05T04:59:50.512+05:30  INFO 38159 --- [           main] org.owasp.webgoat.server.StartWebGoat    : Started StartWebGoat in 1.075 seconds (process running for 1.87)
 __          __       _        _____                   _
 \ \        / /      | |      / ____|                 | |
  \ \  /\  / /  ___  | |__   | |  __    ___     __ _  | |_
   \ \/  \/ /  / _ \ | '_ \  | | |_ |  / _ \   / _' | | __|
    \  /\  /  |  __/ | |_) | | |__| | | (_) | | (_| | | |_
     \/  \/    \___| |_.__/   \_____|  \___/   \__,_|  \__|

2024-09-05T04:59:50.692+05:30  INFO 38159 --- [           main] org.owasp.webgoat.server.StartWebGoat    : No active profile set, falling back to 1 default profile: "defau
lt"
2024-09-05T04:59:51.665+05:30  INFO 38159 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-09-05T04:59:51.733+05:30  INFO 38159 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 61 ms. Found 2 JPA r
epository interfaces.
2024-09-05T04:59:52.141+05:30  WARN 38159 --- [           main] io.undertow.websockets.jsr               : UT026010: Buffer pool was not set on WebSocketDeploymentInfo, th
e default pool will be used
2024-09-05T04:59:52.159+05:30  INFO 38159 --- [           main] io.undertow.servlet                      : Initializing Spring embedded WebApplicationContext
2024-09-05T04:59:52.160+05:30  INFO 38159 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1340 ms
2024-09-05T04:59:52.350+05:30  INFO 38159 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2024-09-05T04:59:52.692+05:30  INFO 38159 --- [           main] com.zaxxer.hikari.pool.PoolBase          : HikariPool-1 - Driver does not support get/set network timeout f
or connections. (feature not supported)
2024-09-05T04:59:52.693+05:30  INFO 38159 --- [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection org.hsqldb.jdbc.JDBCConnection@3
d4d3fe7
2024-09-05T04:59:52.695+05:30  INFO 38159 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2024-09-05T04:59:52.764+05:30  INFO 38159 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2024-09-05T04:59:52.808+05:30  INFO 38159 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.2.13.Final
2024-09-05T04:59:52.811+05:30  INFO 38159 --- [           main] org.hibernate.cfg.Environment            : HHH000406: Using bytecode reflection optimizer
2024-09-05T04:59:53.093+05:30  INFO 38159 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
```



```
nrm@nrm-VirtualBox:~/Downloads$ java -jar webgoat-2023.8.jar
2024-09-24T22:12:44.065+05:30  INFO 6439 --- [           main] org.owasp.webgoat.server.StartWebGoat
 : Starting StartWebGoat v2023.8 using Java 17.0.12 with PID 6439 (/home/nrm/Downloads/webgoat-2023.8.j
ar started by nrm in /home/nrm/Downloads)
2024-09-24T22:12:44.068+05:30  INFO 6439 --- [           main] org.owasp.webgoat.server.StartWebGoat
 : No active profile set, falling back to 1 default profile: "default"
2024-09-24T22:12:44.567+05:30  INFO 6439 --- [           main] org.owasp.webgoat.server.StartWebGoat
 : Started StartWebGoat in 0.952 seconds (process running for 1.873)

    __          __       _        _____                  _
    \ \        / /      | |      / ____|                 | |
     \ \  /\  / /  ___  | |__   | |  __    ___     __ _  | |_
      \ \/  \/ /  / _ \ | '_ \  | | |_ |  / _ \   / _' | | __|
       \  /\  /  |  __/ | |_) | | |__| | | (_) | | (_| | | |_
        \/  \/    \___| |_.__/   \_____|  \___/   \__,_|  \__|

2024-09-24T22:12:44.634+05:30  INFO 6439 --- [           main] org.owasp.webgoat.server.StartWebGoat
 : No active profile set, falling back to 1 default profile: "default"
2024-09-24T22:12:45.663+05:30  INFO 6439 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate
 : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-09-24T22:12:45.788+05:30  INFO 6439 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate
 : Finished Spring Data repository scanning in 115 ms. Found 2 JPA repository interfaces.
2024-09-24T22:12:46.315+05:30  WARN 6439 --- [           main] io.undertow.websockets.jsr
 : UT026010: Buffer pool was not set on WebSocketDeploymentInfo, the default pool will be used
2024-09-24T22:12:46.343+05:30  INFO 6439 --- [           main] io.undertow.servlet
 : Initializing Spring embedded WebApplicationContext
2024-09-24T22:12:46.345+05:30  INFO 6439 --- [           main] w.s.c.ServletWebServerApplicationContext
 : Root WebApplicationContext: initialization completed in 1632 ms
2024-09-24T22:12:46.627+05:30  INFO 6439 --- [           main] com.zaxxer.hikari.HikariDataSource
 : HikariPool-1 - Starting...
```



WebWolf the small helper

WebWolf is a companion application to **WebGoat**, and it is used for testing various security vulnerabilities related to interactions with an external server. While WebGoat teaches you how to exploit vulnerabilities in a web application (like SQL injection, XSS, etc.), WebWolf allows you to practice additional types of attacks, such as sending phishing emails, intercepting HTTP requests, downloading files from malicious links, and other advanced scenarios.

- Hosting a file
- Receiving email
- Landing page for incoming requests

Since you've already installed **WebGoat 2023.8** (webgoat-2023.8.jar), WebWolf is included as part of that installation. You don't need to install WebWolf separately; it runs alongside WebGoat.

WebWolf URL: http://localhost:9090/WebWolf

Changing WebWolf Port

```
java -jar webgoat-2023.8.jar --webwolf.port=9091
```

Perform Basic Vulnerability Analysis:

o Use OWASP ZAP to scan the web application for vulnerabilities.

o Focus on identifying at least one instance each of SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).

Methodology

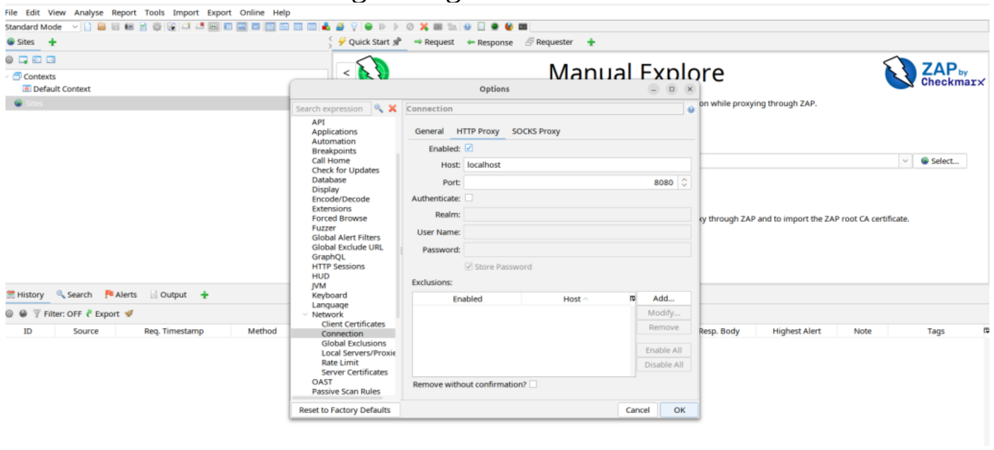
OWASP ZAP Proxy Setup and Configuration

To facilitate the interception, examination, and modification of web requests and responses, OWASP ZAP (Zed Attack Proxy) was set up as a proxy server between the web browser and the internet. This setup allowed for detailed observation and manipulation of traffic to and from WebGoat, which was crucial for identifying and exploiting the vulnerabilities.

Setup Steps:

1. **Download and Install ZAP:**
   - OWASP ZAP was downloaded from the official OWASP website and installed on the local testing machine.
2. **Configure ZAP as a Proxy:**
   - On launching ZAP, the local proxy server was automatically set up. The default configuration listens on port 8080 on the local machine (localhost:8080).

- Manual configuration adjustments were made to ensure ZAP captures traffic from all the browsers used during testing.



Connection

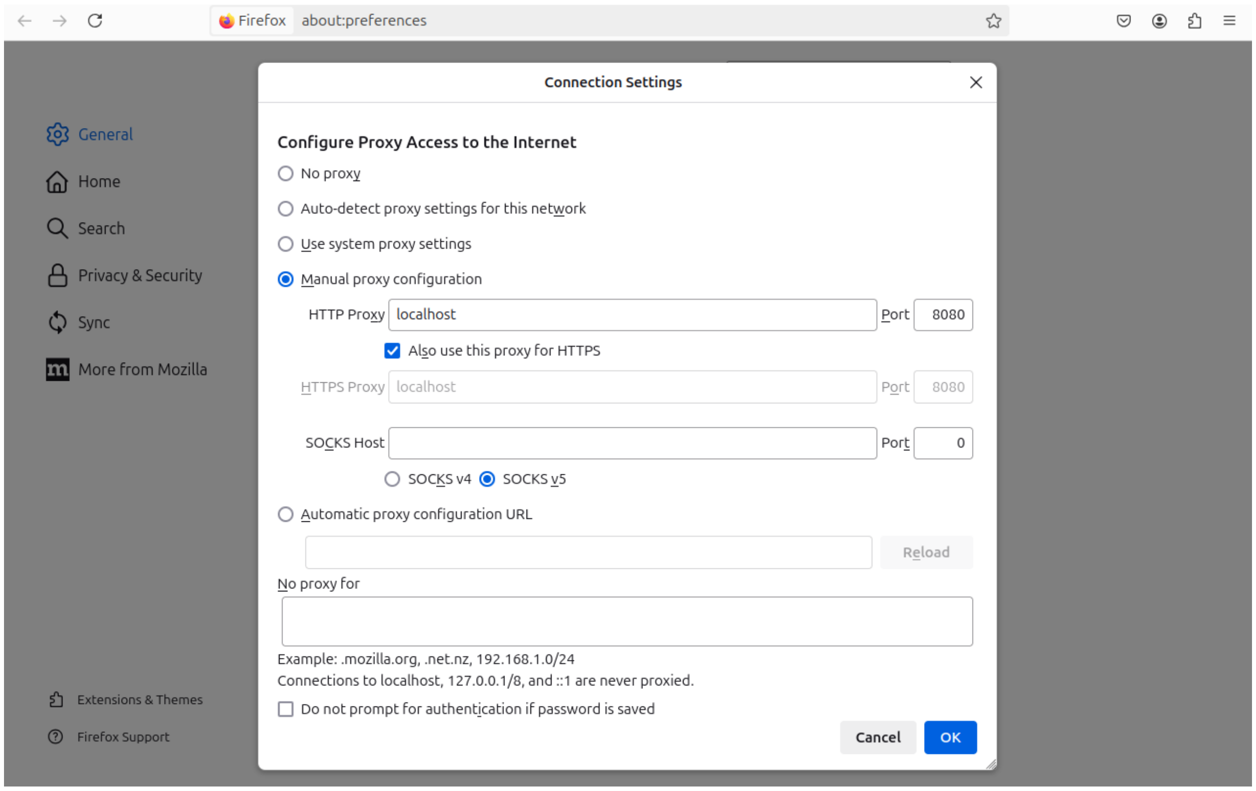
General | HTTP Proxy | SOCKS Proxy

Enabled: ✓
Host: localhost
Port: 8080
Authenticate: ☐
Realm:
User Name:
Password:
✓ Store Password
Exclusions:

Enabled	Host

Remove without confirmation? ☐

3. **Browser Configuration:**
   - The network settings of the browser were configured to use the ZAP proxy by setting the proxy server to localhost and the port to 8080.
   - This configuration directed all browser traffic through ZAP, allowing it to be intercepted and logged.



Connection Settings

Configure Proxy Access to the Internet

○ No proxy
○ Auto-detect proxy settings for this network
○ Use system proxy settings
● Manual proxy configuration

HTTP Proxy: localhost Port: 8080
✓ Also use this proxy for HTTPS
HTTPS Proxy: localhost Port: 8080
SOCKS Host: Port: 0
○ SOCKS v4 ● SOCKS v5

○ Automatic proxy configuration URL

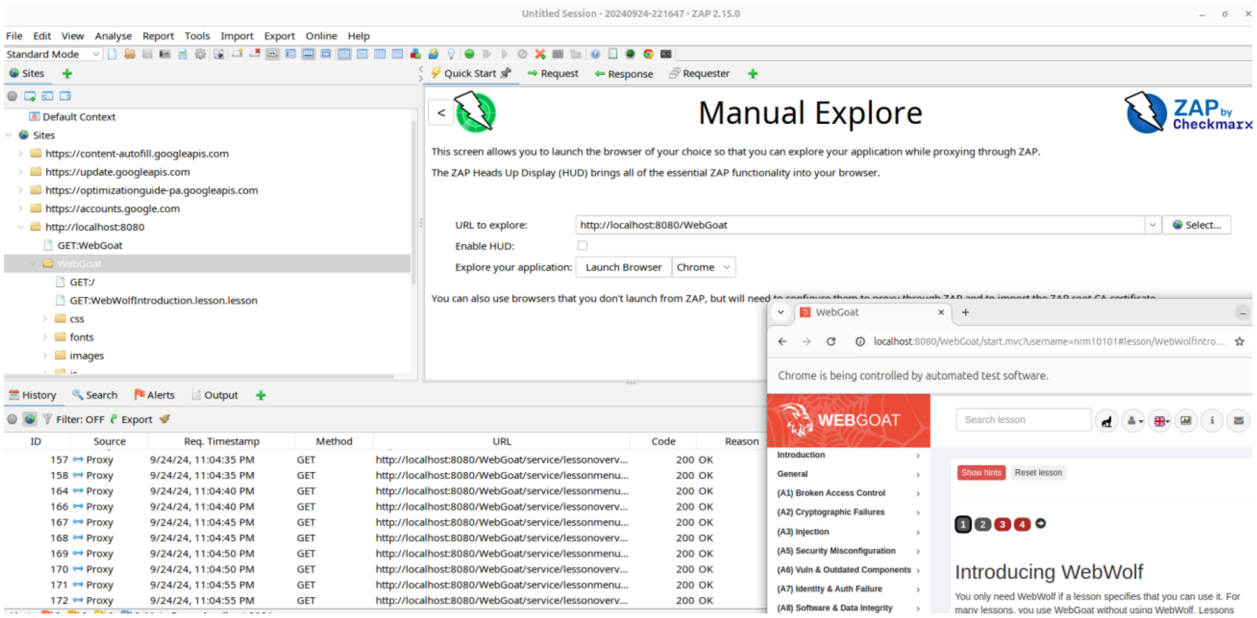
No proxy for

Example: .mozilla.org, .net.nz, 192.168.1.0/24
Connections to localhost, 127.0.0.1/8, and ::1 are never proxied.

☐ Do not prompt for authentication if password is saved

The methodology comprised both automated and manual testing approaches to provide a comprehensive analysis:

1. **Automated Vulnerability Scan:** Using OWASP ZAP to perform an initial scan of the web application to identify potential vulnerabilities.
2. **Manual Testing and Exploitation:** Manually testing and exploiting identified vulnerabilities to understand their impact and document the exploitation process.


Untitled Session - 20240924-221647 - ZAP 2.15.0

Manual Explore

This screen allows you to launch the browser of your choice so that you can explore your application while proxying through ZAP.

The ZAP Heads Up Display (HUD) brings all of the essential ZAP functionality into your browser.

URL to explore: http://localhost:8080/WebGoat

Enable HUD: ☐

Explore your application: Launch Browser | Chrome

You can also use browsers that you don't launch from ZAP, but will need to configure them to proxy through ZAP and to import the ZAP root CA certificate.

Sites:
- Default Context
- Sites
  - https://content-autofill.googleapis.com
  - https://update.googleapis.com
  - https://optimizationguide-pa.googleapis.com
  - https://accounts.google.com
  - http://localhost:8080
    - GET:WebGoat
    - WebGoat
      - GET:/
      - GET:WebWolfIntroduction.lesson.lesson
      - css
      - fonts
      - images

Filter: OFF

ID	Source	Req. Timestamp	Method	URL	Code	Reason
157	Proxy	9/24/24, 11:04:35 PM	GET	http://localhost:8080/WebGoat/service/lessonoverv...	200	OK
158	Proxy	9/24/24, 11:04:35 PM	GET	http://localhost:8080/WebGoat/service/lessonmenu...	200	OK
164	Proxy	9/24/24, 11:04:40 PM	GET	http://localhost:8080/WebGoat/service/lessonmenu...	200	OK
166	Proxy	9/24/24, 11:04:40 PM	GET	http://localhost:8080/WebGoat/service/lessonoverv...	200	OK
167	Proxy	9/24/24, 11:04:45 PM	GET	http://localhost:8080/WebGoat/service/lessonmenu...	200	OK
168	Proxy	9/24/24, 11:04:45 PM	GET	http://localhost:8080/WebGoat/service/lessonoverv...	200	OK
169	Proxy	9/24/24, 11:04:50 PM	GET	http://localhost:8080/WebGoat/service/lessonmenu...	200	OK
170	Proxy	9/24/24, 11:04:50 PM	GET	http://localhost:8080/WebGoat/service/lessonoverv...	200	OK
171	Proxy	9/24/24, 11:04:55 PM	GET	http://localhost:8080/WebGoat/service/lessonmenu...	200	OK
172	Proxy	9/24/24, 11:04:55 PM	GET	http://localhost:8080/WebGoat/service/lessonoverv...	200	OK

localhost:8080/WebGoat/start.mvc?username=nrm10101#lesson/WebWolfIntro...

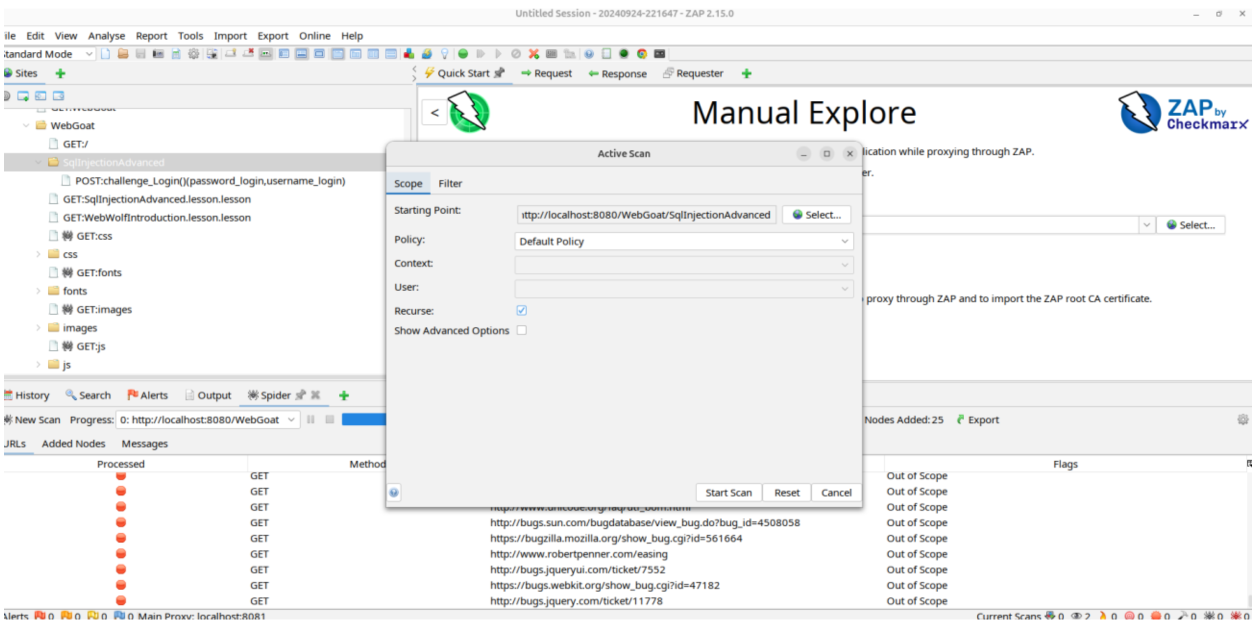
Chrome is being controlled by automated test software.

WEBGOAT: Introduction; General; (A1) Broken Access Control; (A2) Cryptographic Failures; (A3) Injection; (A5) Security Misconfiguration; (A6) Vuln & Outdated Components; (A7) Identity & Auth Failure; (A8) Software & Data Integrity

Introducing WebWolf

You only need WebWolf if a lesson specifies that you can use it. For many lessons, you use WebGoat without using WebWolf. Lessons

Figure 1:Manual Tesing


Untitled Session - 20240924-221647 - ZAP 2.15.0

Manual Explore

Sites:
- WebGoat
  - GET:/
  - SqlInjectionAdvanced
    - POST:challenge_Login()(password_login,username_login)
  - GET:SqlInjectionAdvanced.lesson.lesson
  - GET:WebWolfIntroduction.lesson.lesson
  - GET:css
  - css
  - GET:fonts
  - fonts
  - GET:images
  - images
  - GET:js
  - js

Active Scan

Scope | Filter

Starting Point: http://localhost:8080/WebGoat/SqlInjectionAdvanced

Policy: Default Policy

Context:

User:

Recurse: ☑

Show Advanced Options: ☐

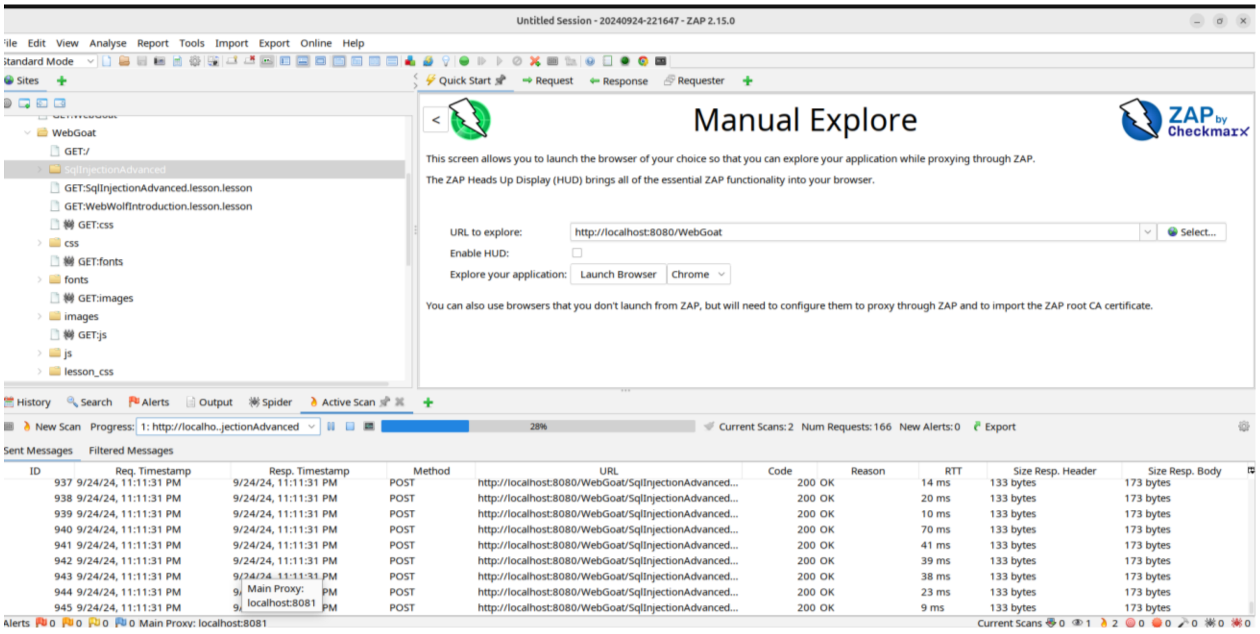
Start Scan | Reset | Cancel

Spider — New Scan — Progress: 0: http://localhost:8080/WebGoat — Nodes Added: 25 — Export

Processed	Method	URL	Flags
🔴	GET		Out of Scope
🔴	GET		Out of Scope
🔴	GET		Out of Scope
🔴	GET	http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4508058	Out of Scope
🔴	GET	https://bugzilla.mozilla.org/show_bug.cgi?id=561664	Out of Scope
🔴	GET	http://www.robertpenner.com/easing	Out of Scope
🔴	GET	http://bugs.jqueryui.com/ticket/7552	Out of Scope
🔴	GET	https://bugs.webkit.org/show_bug.cgi?id=47182	Out of Scope
🔴	GET	http://bugs.jquery.com/ticket/11778	Out of Scope

Alerts 0 0 0 0 Main Proxy: localhost:8081

Figure 2:Active Scan for Advanced Sql injection



This screen allows you to launch the browser of your choice so that you can explore your application while proxying through ZAP.

The ZAP Heads Up Display (HUD) brings all of the essential ZAP functionality into your browser.

URL to explore: http://localhost:8080/WebGoat

Enable HUD: ☐

Explore your application: Launch Browser Chrome

You can also use browsers that you don't launch from ZAP, but will need to configure them to proxy through ZAP and to import the ZAP root CA certificate.

Progress: 1: http://localho..jectionAdvanced 28% Current Scans: 2 Num Requests: 166 New Alerts: 0

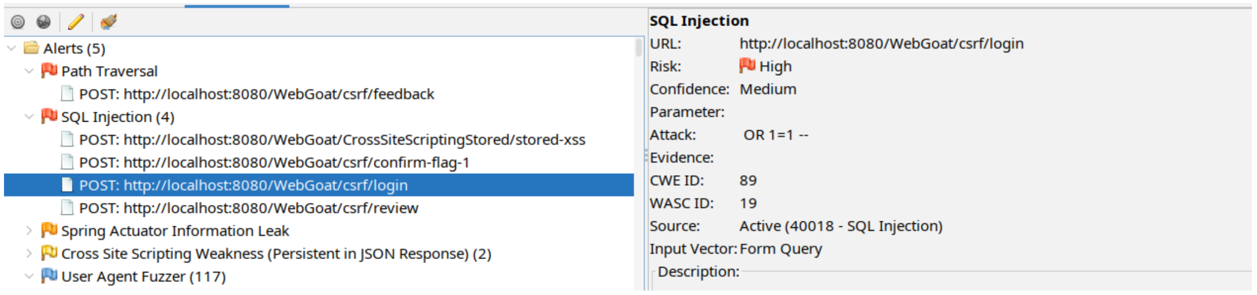
ID	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
937	9/24/24, 11:11:31 PM	9/24/24, 11:11:31 PM	POST	http://localhost:8080/WebGoat/SqlInjectionAdvanced...	200	OK	14 ms	133 bytes	173 bytes
938	9/24/24, 11:11:31 PM	9/24/24, 11:11:31 PM	POST	http://localhost:8080/WebGoat/SqlInjectionAdvanced...	200	OK	20 ms	133 bytes	173 bytes
939	9/24/24, 11:11:31 PM	9/24/24, 11:11:31 PM	POST	http://localhost:8080/WebGoat/SqlInjectionAdvanced...	200	OK	10 ms	133 bytes	173 bytes
940	9/24/24, 11:11:31 PM	9/24/24, 11:11:31 PM	POST	http://localhost:8080/WebGoat/SqlInjectionAdvanced...	200	OK	70 ms	133 bytes	173 bytes
941	9/24/24, 11:11:31 PM	9/24/24, 11:11:31 PM	POST	http://localhost:8080/WebGoat/SqlInjectionAdvanced...	200	OK	41 ms	133 bytes	173 bytes
942	9/24/24, 11:11:31 PM	9/24/24, 11:11:31 PM	POST	http://localhost:8080/WebGoat/SqlInjectionAdvanced...	200	OK	39 ms	133 bytes	173 bytes
943	9/24/24, 11:11:31 PM	9/24/24, 11:11:31 PM	POST	http://localhost:8080/WebGoat/SqlInjectionAdvanced...	200	OK	38 ms	133 bytes	173 bytes
944	9/24/24, 11:11:31 PM	[illegible] PM	POST	http://localhost:8080/WebGoat/SqlInjectionAdvanced...	200	OK	23 ms	133 bytes	173 bytes
945	9/24/24, 11:11:31 PM	[illegible] PM	POST	http://localhost:8080/WebGoat/SqlInjectionAdvanced...	200	OK	9 ms	133 bytes	173 bytes

Alerts 0 0 0 0 Main Proxy: localhost:8081 Current Scans 0 1 2 0 0 0 0 0

Figure 3:Active Scan Advanced Sql injection

Detailed Findings and Analysis

1. SQL Injection

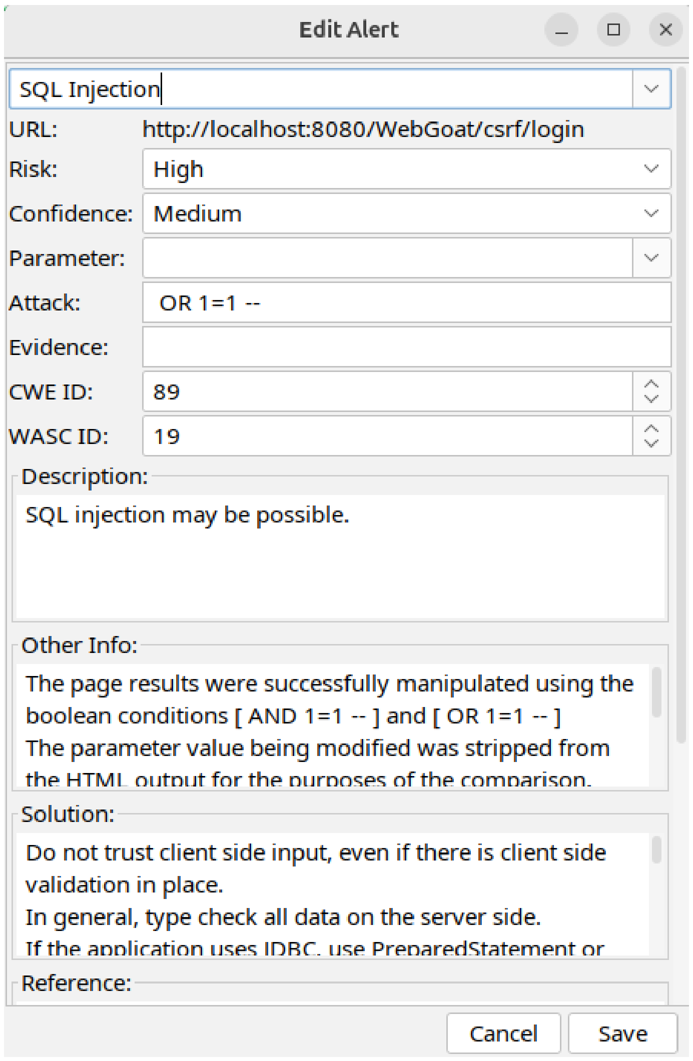


- Alerts (5)
  - Path Traversal
    - POST: http://localhost:8080/WebGoat/csrf/feedback
  - SQL Injection (4)
    - POST: http://localhost:8080/WebGoat/CrossSiteScriptingStored/stored-xss
    - POST: http://localhost:8080/WebGoat/csrf/confirm-flag-1
    - POST: http://localhost:8080/WebGoat/csrf/login
    - POST: http://localhost:8080/WebGoat/csrf/review
  - Spring Actuator Information Leak
  - Cross Site Scripting Weakness (Persistent in JSON Response) (2)
  - User Agent Fuzzer (117)

SQL Injection

URL:	http://localhost:8080/WebGoat/csrf/login
Risk:	High
Confidence:	Medium
Parameter:	
Attack:	OR 1=1 --
Evidence:	
CWE ID:	89
WASC ID:	19
Source:	Active (40018 - SQL Injection)
Input Vector:	Form Query
Description:	

- **Vulnerability Details:**
  - **Location:** CrossSiteScriptingStored/stored-xss
  - **Description:** The application accepts unfiltered user input in the text parameter, which is directly used in SQL queries, enabling SQL code execution.
  - **Risk Level:** High
  - **Confidence:** Medium
  - **CWE ID:** 89



SQL Injection

URL: http://localhost:8080/WebGoat/csrf/login

Risk: High

Confidence: Medium

Parameter:

Attack: OR 1=1 --

Evidence:

CWE ID: 89

WASC ID: 19

Description:

SQL injection may be possible.

Other Info:

The page results were successfully manipulated using the boolean conditions [AND 1=1 --] and [OR 1=1 --]
The parameter value being modified was stripped from the HTML output for the purposes of the comparison.

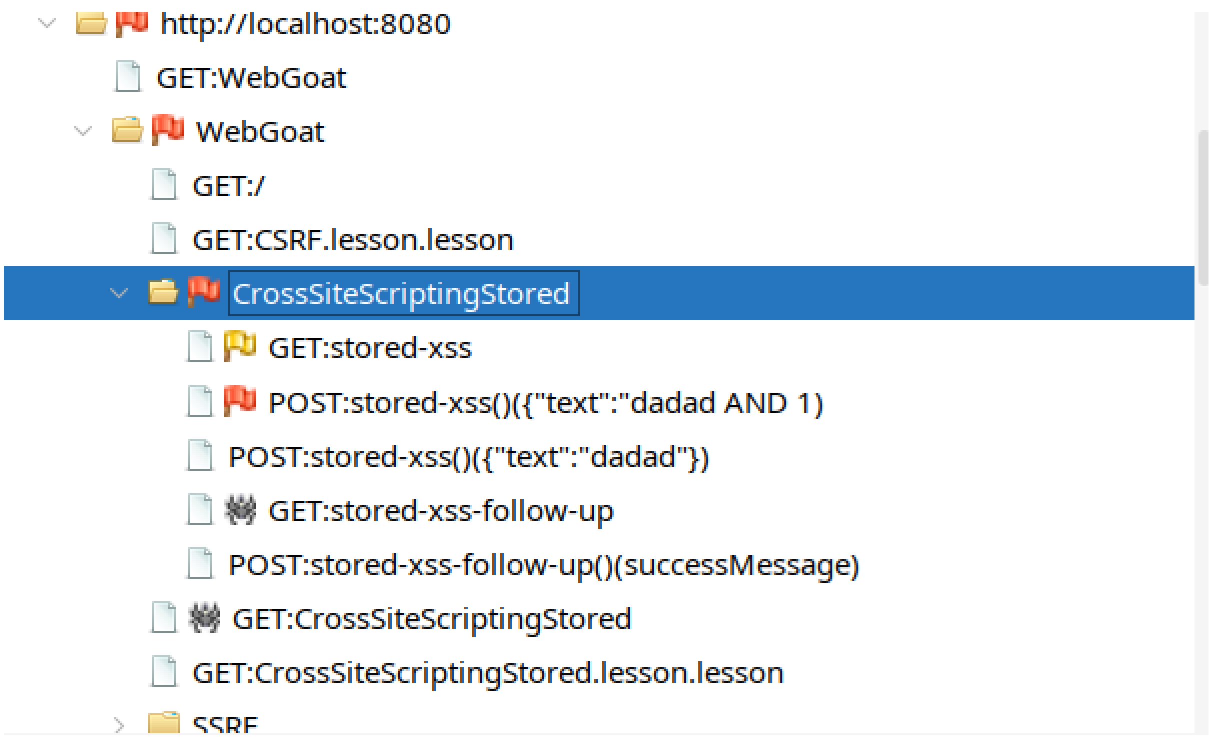
Solution:

Do not trust client side input, even if there is client side validation in place.
In general, type check all data on the server side.
If the application uses JDBC, use PreparedStatement or

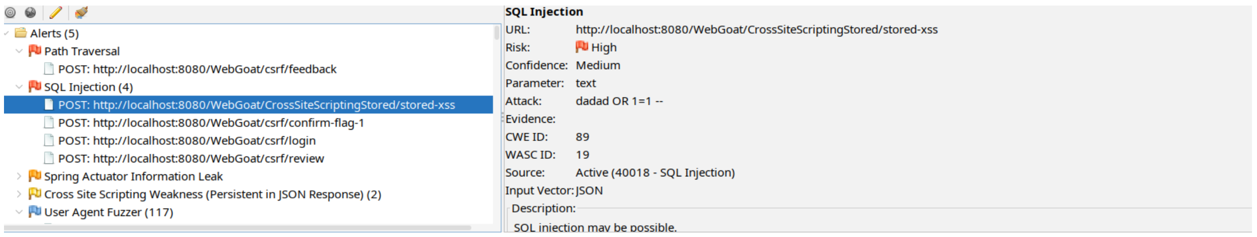
Reference:

- **Exploitation Technique:**
  - A crafted SQL payload (dadad OR 1=1--) was inserted into the input field, which manipulated the SQL query to return all entries, bypassing intended query logic.
  - **Screenshot:** [Include a screenshot of the SQL Injection in action]
- **Impact Assessment:**
  - This vulnerability could allow unauthorized access to sensitive database content and lead to data theft or manipulation.
- **Mitigation Recommendations:**
  - Use prepared statements with parameterized queries.
  - Validate and sanitize all user inputs on the server side.
  - Regular security audits of code handling user inputs.

2. **Stored Cross-Site Scripting (XSS)**



```
http://localhost:8080
    GET:WebGoat
    WebGoat
        GET:/
        GET:CSRF.lesson.lesson
        CrossSiteScriptingStored
            GET:stored-xss
            POST:stored-xss()({"text":"dadad AND 1)
            POST:stored-xss()({"text":"dadad"})
            GET:stored-xss-follow-up
            POST:stored-xss-follow-up()(successMessage)
        GET:CrossSiteScriptingStored
        GET:CrossSiteScriptingStored.lesson.lesson
        SSRF
```

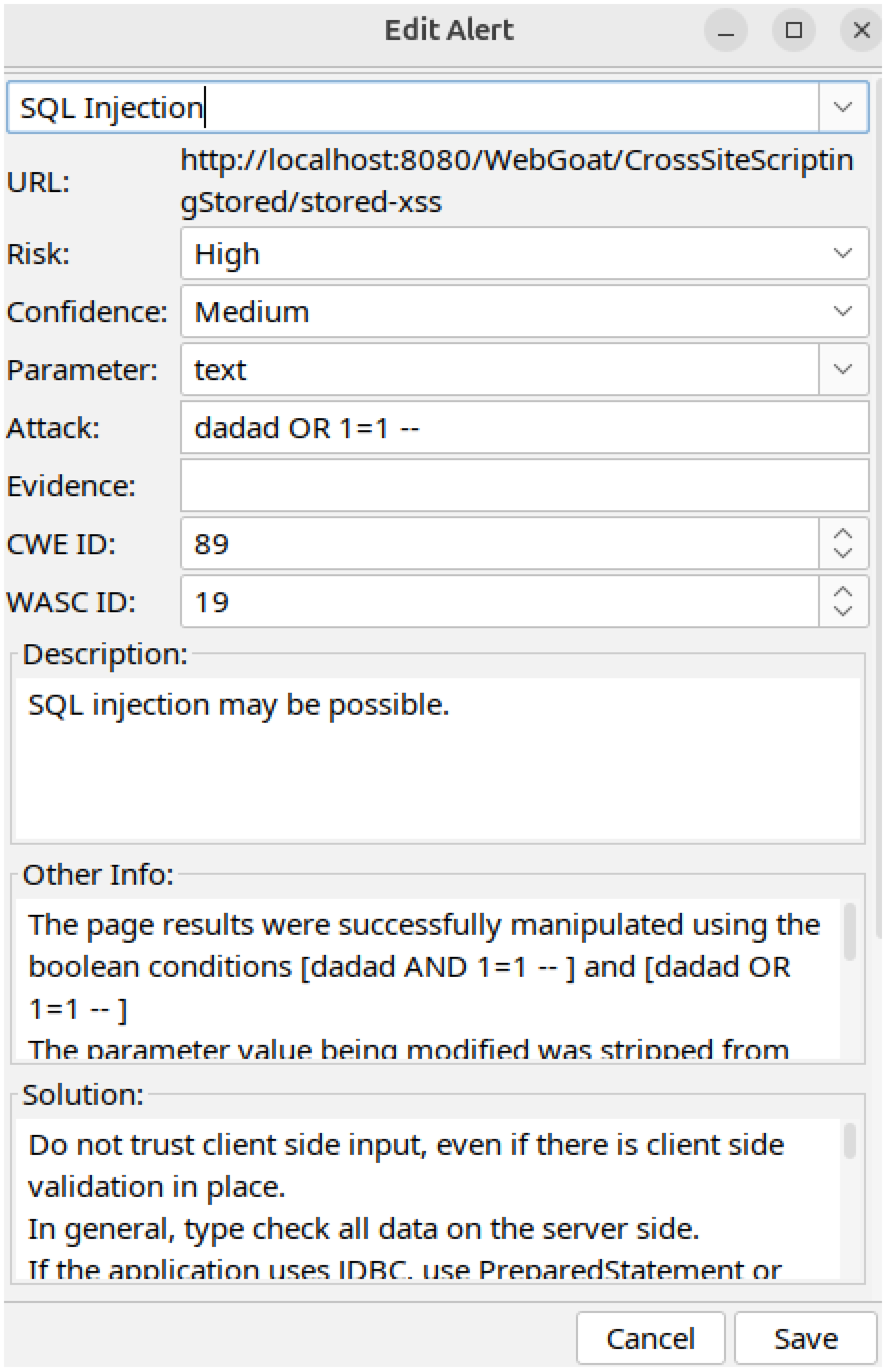


```
Alerts (5)
  Path Traversal
    POST: http://localhost:8080/WebGoat/csrf/feedback
  SQL Injection (4)
    POST: http://localhost:8080/WebGoat/CrossSiteScriptingStored/stored-xss
    POST: http://localhost:8080/WebGoat/csrf/confirm-flag-1
    POST: http://localhost:8080/WebGoat/csrf/login
    POST: http://localhost:8080/WebGoat/csrf/review
  Spring Actuator Information Leak
  Cross Site Scripting Weakness (Persistent in JSON Response) (2)
  User Agent Fuzzer (117)
```

```
SQL Injection
URL:          http://localhost:8080/WebGoat/CrossSiteScriptingStored/stored-xss
Risk:         High
Confidence:   Medium
Parameter:    text
Attack:       dadad OR 1=1 --
Evidence:
CWE ID:       89
WASC ID:      19
Source:       Active (40018 - SQL Injection)
Input Vector: JSON
Description:
SQL injection may be possible.
```

- **Location:** `CrossSiteScriptingStored/stored-xss`
- **Vulnerability Details:**

  - **Description:** The SQL Injection vulnerability is present in the text input field of the stored XSS page, where user inputs are directly incorporated into SQL commands without proper sanitization or validation.
  - **Risk Level:** High
  - **Confidence:** Medium
  - **CWE ID:** 89 (SQL Injection)
  - **WASC ID:** 19


Edit Alert

SQL Injection

URL:	http://localhost:8080/WebGoat/CrossSiteScriptingStored/stored-xss
Risk:	High
Confidence:	Medium
Parameter:	text
Attack:	dadad OR 1=1 --
Evidence:	
CWE ID:	89
WASC ID:	19

Description:

SQL injection may be possible.

Other Info:

The page results were successfully manipulated using the boolean conditions [dadad AND 1=1 --] and [dadad OR 1=1 --]
The parameter value being modified was stripped from

Solution:

Do not trust client side input, even if there is client side validation in place.
In general, type check all data on the server side.
If the application uses JDBC, use PreparedStatement or

- **Technical Insights:**
    - The vulnerability arises from the application's handling of the 'text' parameter within SQL queries. Specifically, input values such as `dadad OR 1=1 --` demonstrate that the application is vulnerable to SQL manipulation through boolean-based SQL Injection attacks.
    - **Affected Parameter:** `text`
    - **Input Vector:** JSON

  - **Evidence:** Successful manipulation of SQL queries using boolean conditions `dadad AND 1=1 --` and `dadad OR 1=1 --`.

- **Exploitation Technique:**

  - By injecting SQL control characters and logic-altering syntax (`OR 1=1`), an attacker can alter the query to return more data than intended or bypass authentication mechanisms.
  - **Example Payload:** `dadad' OR '1'='1 --`
  - This results in unauthorized data disclosure, unauthorized data manipulation, or complete bypass of authentication.

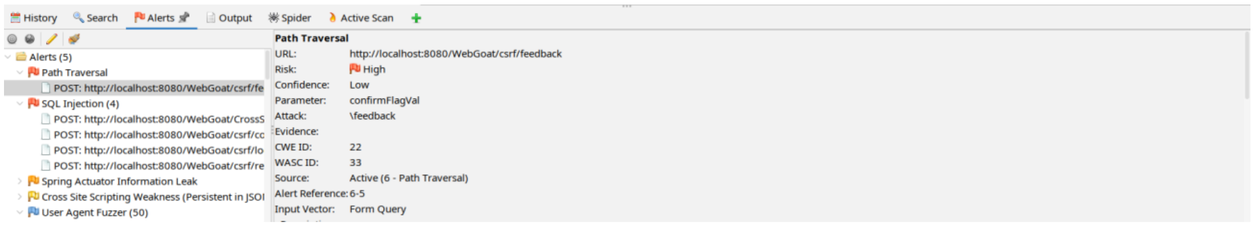
- **Impact Assessment:**

  - SQL Injection can lead to unauthorized access to sensitive database information, data corruption, and potential system compromise.
  - This vulnerability may allow attackers to extract sensitive data, execute administrative operations, or even execute arbitrary code on the server depending on the database configuration and permissions.

- **Mitigation Recommendations:**

  - **Parameterized Queries:** Utilize parameterized queries with prepared statements to ensure that SQL code cannot be injected through user input.
  - **Escaping All User Inputs:** Implement proper escaping routines for all user inputs, especially those used within SQL queries.
  - **Use of Stored Procedures:** Encourage the use of stored procedures to handle database transactions, ensuring that any dynamic SQL generation is handled securely without direct input from untrusted sources.
  - **Least Privilege:** Ensure that database operations are executed with the minimum privileges necessary to accomplish the task. Avoid using highly privileged accounts like 'sa' or 'db-owner' for application database access.
  - **Input Validation:** Implement strict input validation using allowlists to restrict allowed characters, preventing the inclusion of SQL syntax.
  - **Security Audits and Code Reviews:** Regularly conduct security audits and code reviews to identify and rectify potential vulnerabilities in the handling of user inputs and SQL query generation.

3. **Path Traversal**

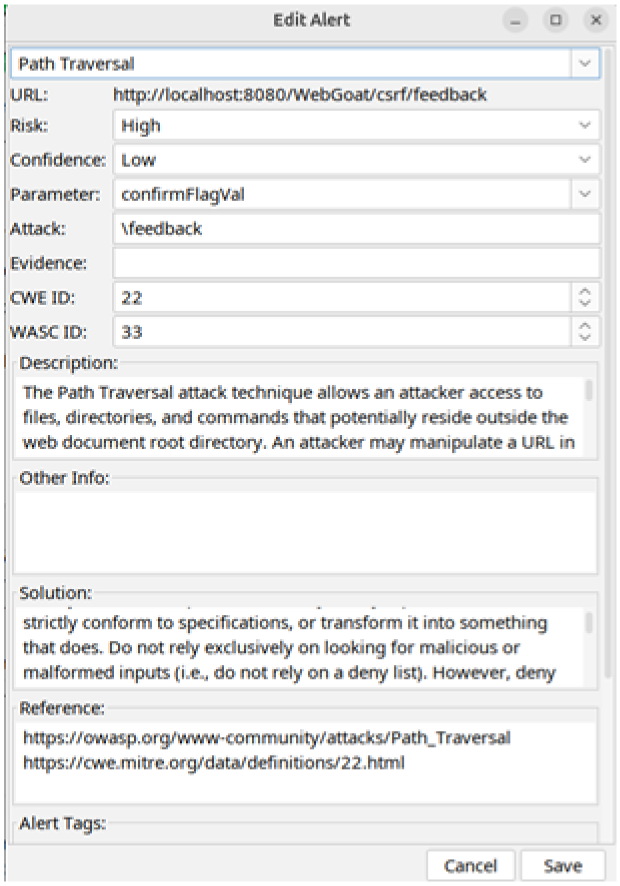


Alerts (5)
- Path Traversal
  - POST: http://localhost:8080/WebGoat/csrf/fe
- SQL Injection (4)
  - POST: http://localhost:8080/WebGoat/CrossS
  - POST: http://localhost:8080/WebGoat/csrf/cc
  - POST: http://localhost:8080/WebGoat/csrf/lo
  - POST: http://localhost:8080/WebGoat/csrf/re
- Spring Actuator Information Leak
- Cross Site Scripting Weakness (Persistent in JSOI
- User Agent Fuzzer (50)

Path Traversal

URL:	http://localhost:8080/WebGoat/csrf/feedback
Risk:	High
Confidence:	Low
Parameter:	confirmFlagVal
Attack:	\feedback
Evidence:	
CWE ID:	22
WASC ID:	33
Source:	Active (6 - Path Traversal)
Alert Reference:	6-5
Input Vector:	Form Query

- **Location:** csrf/feedback
- **Vulnerability Details:**
  - **Description:** The Path Traversal vulnerability in the csrf/feedback endpoint allows unauthorized access to files and directories outside of the web document root directory. The vulnerability stems from inadequate validation of user-supplied input, which is used to construct file paths.
  - **Parameter Affected:** confirmFlagVal
  - **Risk Level:** High
  - **Confidence:** Low
  - **CWE ID:** 22 (Improper Limitation of a Pathname to a Restricted Directory)
  - **WASC ID:** 33



Path Traversal

URL:	http://localhost:8080/WebGoat/csrf/feedback
Risk:	High
Confidence:	Low
Parameter:	confirmFlagVal
Attack:	\feedback
Evidence:	
CWE ID:	22
WASC ID:	33

Description:
The Path Traversal attack technique allows an attacker access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in

Other Info:

Solution:
strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny

Reference:
https://owasp.org/www-community/attacks/Path_Traversal
https://cwe.mitre.org/data/definitions/22.html

Alert Tags:

- **Technical Insights:**

    - Attackers exploit this vulnerability by manipulating input parameters with sequences like "../" to navigate the filesystem, potentially accessing sensitive files or executing commands.
    - The input vfeedback could be manipulated to include sequences such as "..%u2216", "..%c0%af", or "..%2e%2e%2f" to bypass basic filters and access critical system files.
- **Exploitation Technique:**
    - An attacker crafts a URL or form input that includes directory traversal sequences to access or manipulate files outside the intended directory.
    - **Example Payload:** ../../etc/passwd
    - This allows the attacker to read sensitive information from system files, which could lead to further exploitation of the system.
- **Impact Assessment:**
    - Successful exploitation can lead to unauthorized disclosure of sensitive data, unauthorized access to the application data, or server compromise.
    - Potential for system takeover if critical system files or configurations are accessed or modified.
- **Mitigation Recommendations:**
    - **Input Validation:** Implement rigorous validation of all user inputs. Use an allowlist approach where only known good inputs that conform strictly to specifications are accepted. Inputs should be checked for type, length, format, and range.
    - **Filesystem Access Controls:** Restrict filesystem access to only those directories and files that the application strictly requires. Employ least privilege principles.
    - **Sanitization and Canonicalization:** Inputs used in file paths should be decoded, validated, sanitized, and canonicalized before use. Utilize a built-in path canonicalization function like realpath() in PHP or similar functions in other programming languages to remove ".." sequences and symbolic links.
    - **Security Configuration:** Configure the server and application environment to operate with minimal permissions. Run application processes in a sandboxed or jailed environment to limit the impact of a potential exploit.
    - **Error Handling:** Ensure that error messages do not disclose information about the filesystem or provide clues that could assist an attacker.
    - **Logging and Monitoring:** Implement comprehensive logging of file access attempts and regular monitoring for unusual access patterns that could indicate an attempt to exploit this vulnerability.

Further Actions:

- Regular security audits and penetration testing should be performed to identify and mitigate new vulnerabilities related to path traversal and other potential security threats.
- Continuous education and training for developers on secure coding practices focusing on handling user input and file manipulation securely.

Security Best Practices and Recommendations

Input Validation

- **Implementation Strategy:** Employ comprehensive validation on all user inputs, using both server-side and client-side validation to create a defense-in-depth approach. Inputs should be checked against a strict set of rules, such as type, length, format, and range.
- **Tools and Techniques:** Utilize frameworks that automatically enforce these checks and provide additional protection layers like escaping outputs to prevent injection attacks.

Regular Updates

- **Update Policy:** Establish a routine for regularly updating all software components used within the application. This includes the main application platform, any libraries, third-party plugins, and development tools.
- **Automation:** Implement automated tools to track outdated components or known vulnerabilities using services like Dependabot or OWASP Dependency-Check.

Security Training

- **Developer Training:** Conduct regular training sessions for developers focused on secure coding practices, common vulnerabilities specific to the technologies in use, and general security awareness.
- **Resources and Tools:** Provide access to up-to-date security resources, coding guidelines, and tools that can aid in the development of secure applications. Encourage participation in security forums and workshops.

Security Testing

- **Integrating Security into SDLC:** Embed security testing within the Software Development Life Cycle (SDLC). This includes:
    - **Static Application Security Testing (SAST):** Analyze source code for security vulnerabilities early in the development stages.
    - **Dynamic Application Security Testing (DAST):** Conduct tests on a running application to simulate real-world hacking attacks and identify vulnerabilities.
    - **Penetration Testing:** Regularly schedule manual penetration testing conducted by experienced security professionals.
- **Continuous Monitoring:** Set up continuous monitoring tools to detect and alert on security anomalies in real-time.

Conclusion

This detailed analysis emphasizes the critical role of proactive and comprehensive security practices in web application development. By understanding common vulnerabilities and integrating robust mitigation strategies, developers can significantly bolster the security and integrity of their applications. Such measures not only protect applications from emerging threats but also build trust with end-users and safeguard organizational reputations. As cyber threats evolve, maintaining a proactive security posture will be indispensable in defending against potential breaches and ensuring compliance with international security standards.

Future Outlook

- **Adoption of AI in Security:** Explore the integration of artificial intelligence and machine learning techniques to predict and mitigate potential threats before they exploit vulnerabilities in applications.
- **Enhanced DevSecOps Practices:** Strengthen the collaboration between development, security, and operations teams to ensure seamless security integration across all phases of application development.

By adopting these enhanced practices and staying informed about the latest security trends, organizations can ensure that they are well-prepared to face the challenges of modern web application security.