JAN -2025

To

FEB -2025

Introduction To Web Application Security

- By : KISHORI KHATKE

- To : THE RED USERS

TASK 2

OBJECTIVE:

The primary goal of this task is to deepen your understanding of common web application vulnerabilities by studying a simple, intentionally vulnerable web application. By doing so, you'll gain valuable insights into how attackers exploit these weaknesses and learn the fundamental principles of securing web applications against such threats.

Common Web Application Vulnerabilities:

SQL Injection (SQLi): A vulnerability that allows attackers to manipulate SQL queries by injecting malicious code into input fields. This can lead to unauthorized access to the database, data leakage, or even deletion of data.

Cross-Site Scripting (XSS): This vulnerability occurs when an application allows users to inject malicious scripts into web pages viewed by others. These scripts can be used to steal cookies, session tokens, or other sensitive information.

Cross-Site Request Forgery (CSRF): An attack where a malicious actor tricks a user into performing actions on a web application without their consent. This can result in unauthorized transactions or changes to user data.

Understanding Attack Methods:

How Attackers Exploit Vulnerabilities: By analyzing a vulnerable web application, you can see firsthand how attackers find and exploit weaknesses. This understanding is crucial for developing effective countermeasures.

Techniques Used by Attackers: Common techniques include input manipulation, code injection, session hijacking, and exploiting insecure communication channels.

The Role of Security Measures:

Preventative Measures: Implementing security practices such as input validation, output encoding, and secure session management to prevent vulnerabilities from being exploited.

Detective Measures: Using tools like OWASP ZAP to scan applications for vulnerabilities and regularly monitoring logs for suspicious activities.

Impact of Vulnerabilities:

Potential Damage: Unaddressed vulnerabilities can lead to significant security breaches, including data theft, financial losses, and damage to organizational reputation.

Real-World Examples: Studying past incidents where vulnerabilities were exploited can provide insights into the severity and consequences of these security flaws.

· **Benefits of Learning About Vulnerabilities**:·

Proactive Defense: By understanding common vulnerabilities and how they are exploited, you can take proactive steps to secure web applications.

Skill Development: This task will enhance your cybersecurity skills, making you more adept at identifying and mitigating security risks.

WEB APPLICATION SECURITY - WHAT IS IT, HOW IT IS USED ?

Web Application Security

Web application security involves implementing strategies and measures to protect web applications from cyber threats and vulnerabilities. As web applications become increasingly integral to business operations and personal use, ensuring their security is critical for safeguarding sensitive data, maintaining user trust, and complying with regulatory requirements.

Key Aspects of Web Application Security:

Protection Against Common Vulnerabilities:

SQL Injection: This involves attackers injecting malicious SQL code into input fields to manipulate database queries, leading to unauthorized access to data.

Cross-Site Scripting (XSS): Attackers inject malicious scripts into web pages viewed by others, potentially stealing cookies, session tokens, or other sensitive information.

Cross-Site Request Forgery (CSRF): Attackers trick users into performing actions they didn't intend to by exploiting their authenticated sessions with a trusted website.

Importance of Secure Development Practices:

Input Validation: Ensuring that all input data is validated to prevent injection attacks and other forms of malicious data manipulation.

Output Encoding: Encoding data before rendering it to users to prevent XSS attacks.

Secure Authentication and Authorization: Implementing robust methods for verifying user identities and controlling access to resources.

Encryption:

Data in Transit: Using HTTPS to encrypt data transmitted between clients and servers, protecting it from interception and tampering.

Data at Rest: Encrypting sensitive data stored within databases to safeguard it from unauthorized access, even if the storage medium is compromised.

Regular Security Testing and Monitoring:

Vulnerability Scanning: Using tools like OWASP ZAP to identify potential security flaws within web applications.

Penetration Testing: Conducting regular penetration tests to simulate real-world attacks and identify weaknesses before attackers can exploit them.

Monitoring and Logging: Keeping an eye on application logs and monitoring network traffic to detect unusual or suspicious activities.

Adopting Security Frameworks and Standards:

OWASP Top Ten: Following the guidelines provided by the Open Web Application Security Project (OWASP) to address the most critical security risks to web applications.

Secure Development Lifecycle (SDL): Integrating security practices throughout the software development lifecycle, from design to deployment and maintenance.

Educating Developers and Users:

Training: Regularly training developers on secure coding practices to minimize the introduction of vulnerabilities during development.

User Awareness: Educating users about best practices for maintaining their security, such as recognizing phishing attempts and using strong passwords.

In essence, web application security is a continuous process that involves proactive measures to identify and address potential vulnerabilities, regular testing to detect new threats, and ongoing education to ensure that both developers and users are aware of the latest security practices. By prioritizing security, organizations can protect their web applications from a wide range of cyber threats and ensure the confidentiality, integrity, and availability of their data.

THE PROJECT’S GOAL

This project's main goal is to use practical experience to examine and comprehend typical web application vulnerabilities. The project offers a controlled environment for identifying, exploiting, and documenting vulnerabilities like SQL Injection (SQLi), Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) by utilizing purposefully susceptible web apps. Developing manual testing skills to identify and exploit vulnerabilities; introducing security analysts to automated scanning tools such as OWASP ZAP; and highlighting the significance of secure coding practices by suggesting efficient mitigation techniques are the objectives of this project. The ultimate objective is to close the gap between web application security theory and practical applications. Utilized Tools and Applications The following resources and programs were utilized in order to accomplish the project's goals:

TOOLS AND TECHNIQUES USED IN THIS PROJECT

In order to achieve the goals of this project, the following tools and applications were utilized:

WebGoat (or DVWA):

Objective: WebGoat and DVWA (Damn Vulnerable Web Application) are intentionally vulnerable web applications created to educate users about web application security. They offer a controlled setting for practicing the identification and exploitation of security vulnerabilities.
Application: Function as the main platform for conducting vulnerability assessments and exploitations.

OWASP ZAP (Zed Attack Proxy):

Objective: OWASP ZAP is an open-source tool aimed at detecting vulnerabilities in web applications. It automates the identification of security issues, including SQL Injection, XSS, and CSRF.
Application: Conducts both automated scans and manual testing to reveal security weaknesses within the web application.

Kali Linux:

Objective: Kali Linux is a specialized Linux distribution designed for penetration testing and ethical hacking, encompassing a comprehensive suite of tools for performing security evaluations.
Application: Serves as the primary operating system for executing the essential tools and utilities required for the project.

Firefox:

Objective: Firefox is a web browser configured to work with OWASP ZAP as a proxy, enabling the interception and examination of web traffic.
Application: Facilitates a thorough analysis of HTTP requests and responses during the vulnerability assessment process.

Command-line:

Objective: These utilities are employed for managing application servers, establishing databases, and analyzing log files.
Application: Assist in various tasks related to configuring the testing environment and evaluating the outcomes.

SYNOPSIS OF VULNERABILITY SCANNING AND ANALYSIS

A online application's security flaws are methodically found, exploited, and documented by the vulnerability analysis effort. This systematic methodology aims to fully comprehend potential security vulnerabilities and create efficient mitigation techniques. This is a thorough explanation of the procedure:

1. Configuration and Setup

- Setting up the environment and installing it:

 - Installing Kali Linux: Install Kali Linux, a specialized distribution designed for security evaluations and penetration testing.

 - Web Application Setup: To establish a controlled testing environment, install and set up a purposefully vulnerable web application on your local computer, such as WebGoat or DVWA (Damn Vulnerable Web Application).

Setting Up the Tool:

- OWASP ZAP: Verify the proper installation and configuration of OWASP ZAP (Zed Attack Proxy). Both automatic scanning and human testing of web application vulnerabilities require this tool.

- Configure a web browser (such as Firefox) to utilize OWASP ZAP as a proxy so that web traffic may be examined in detail.

2. Initiation of Vulnerability scanning:

Executing OWASP ZAP: Use OWASP ZAP to do a preliminary automated scan of the web application in order to find common vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).

Examining Scan Outputs: To identify certain vulnerabilities, carefully examine the scan results. Consider each highlighted issue's significance and seriousness.

Understanding Vulnerabilities: To comprehend the characteristics and possible ramifications of each vulnerability, make use of OWASP ZAP's thorough descriptions and classifications.

3. Testing along with Vulnerability Exploitation

Validation of Vulnerabilities:

Verification of Findings: Conduct manual validation of the vulnerabilities identified by OWASP ZAP. Confirm their authenticity and evaluate their potential for exploitation.

Exploitation Techniques:

- SQL Injection: Execute SQL commands within input fields to obtain unauthorized access to database information.

- Cross-Site Scripting (XSS): Embed harmful JavaScript into input fields to alter the behavior of the web application and retrieve sensitive information.

- Cross-Site Request Forgery (CSRF): Carry out actions on behalf of a user without their approval to illustrate how attackers can take advantage of session vulnerabilities.

Impact Assessment:

Evaluating Consequences: Analyze the potential repercussions of each vulnerability if exploited. This encompasses data breaches, unauthorized access, and disruption of services.

4. Documentation

Comprehensive Recording:

Discovery Methods: Record the procedures followed to identify each vulnerability, detailing the specific tools and techniques employed.

Exploitation Process: Offer a detailed account of the exploitation of each vulnerability, including screenshots and technical specifications.

5. Strategies for Mitigation:

Risk Mitigation: Provide workable and efficient solutions to lessen the vulnerabilities that have been found. Enforcing robust authentication procedures, employing secure communication protocols, and putting input validation into practice could all be part of this.

Provide suggestions for continuing security procedures, such as frequent security audits and constant monitoring, to stop similar vulnerabilities in the future.

In-depth Report:

Compilation: Put all of the information together in a thorough report that explains the vulnerabilities, their effects, and the suggested countermeasures.

Clarity and Precision: Make sure the report is understandable to stakeholders who are technical and those who are not.

> By following this structured approach, the vulnerability analysis project aims to enhance your practical understanding of web application security. This hands-on experience will equip you with the skills needed to identify, exploit, and mitigate web application vulnerabilities effectively, bridging the gap between theoretical knowledge and real-world application.

IMPLEMENTATION OF WEB APPLICATION - WEB GOAT INSTALLATION

To simulate real-world web vulnerabilities, we installed the WebGoat application on a local machine using Kali Linux. The following detailed steps outline the installation and setup process:

Install Prerequisites

Update the System:

Start by updating the package list and upgrading installed packages to ensure your system is up-to-date. Open the terminal and run:

-> sudo apt update && sudo apt upgrade

Install Java Development Kit (JDK):

WebGoat requires Java to run. Install the OpenJDK package by executing:

-> sudo apt install openjdk-11-jdk

Verify Java Installation:

Confirm that Java is installed correctly by checking its version. Run the following command:

-> java --version

You should see output indicating the installed Java version.

Download WebGoat

Clone the WebGoat Repository:

Use the git command to clone the WebGoat repository from GitHub. In your terminal, execute:

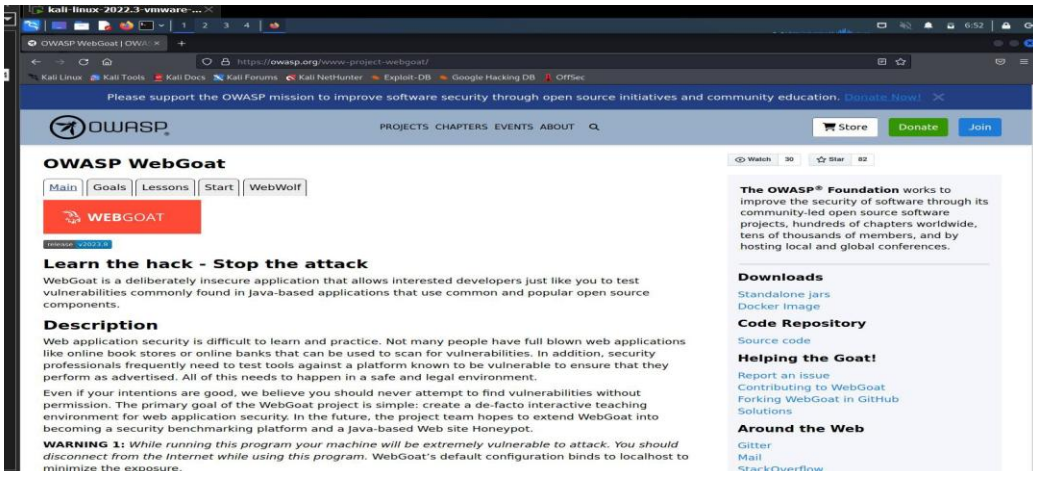
-> git clone https://github.com/WebGoat/WebGoat.git

This command will download all the necessary files into a directory named WebGoat.

Navigate to the WebGoat Directory:

Change your working directory to the newly created WebGoat folder:

-> cd WebGoat



Please support the OWASP mission to improve software security through open source initiatives and community education. Donate Now!

OWASP WebGoat

Learn the hack - Stop the attack

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.

Description

Web application security is difficult to learn and practice. Not many people have full blown web applications like online book stores or online banks that can be used to scan for vulnerabilities. In addition, security professionals frequently need to test tools against a platform known to be vulnerable to ensure that they perform as advertised. All of this needs to happen in a safe and legal environment.

Even if your intentions are good, we believe you should never attempt to find vulnerabilities without permission. The primary goal of the WebGoat project is simple: create a de-facto interactive teaching environment for web application security. In the future, the project team hopes to extend WebGoat into becoming a security benchmarking platform and a Java-based Web site Honeypot.

WARNING 1: *While running this program your machine will be extremely vulnerable to attack. You should disconnect from the Internet while using this program.* WebGoat's default configuration binds to localhost to minimize the exposure.

The OWASP® Foundation works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Downloads

Standalone jars
Docker Image

Code Repository

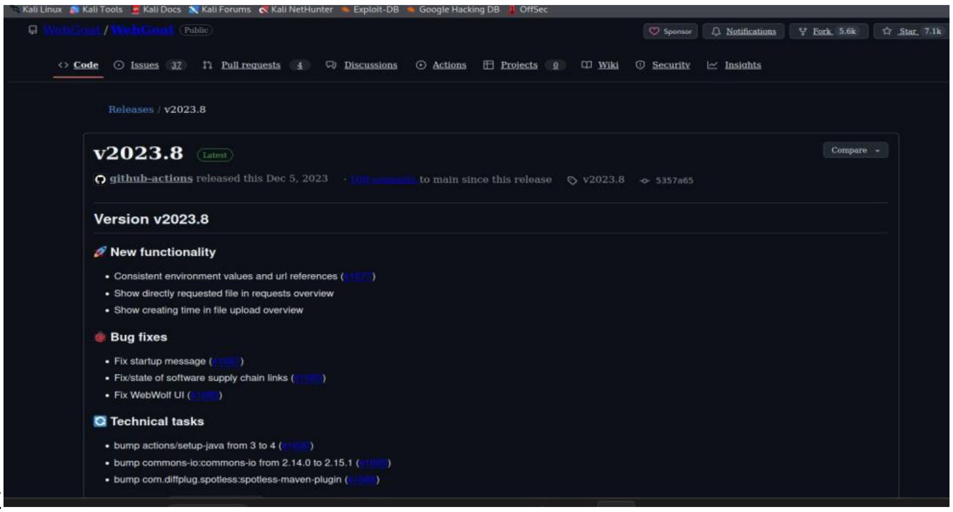
Source code

Helping the Goat!

Report an issue
Contributing to WebGoat
Forking WebGoat in GitHub
Solutions

Around the Web

Gitter
Mail
StackOverflow



Releases / v2023.8

v2023.8 Latest

github-actions released this Dec 5, 2023 · 108 commits to main since this release v2023.8 5357a65

Version v2023.8

New functionality

- Consistent environment values and url references ([illegible])
- Show directly requested file in requests overview
- Show creating time in file upload overview

Bug fixes

- Fix startup message ([illegible])
- Fix/state of software supply chain links ([illegible])
- Fix WebWolf UI ([illegible])

Technical tasks

- bump actions/setup-java from 3 to 4 ([illegible])
- bump commons-io:commons-io from 2.14.0 to 2.15.1 ([illegible])
- bump com.diffplug.spotless:spotless-maven-plugin ([illegible])

Run WebGoat

Build the Project (If Necessary):

If the project needs to be built, navigate to the project directory and use Maven to build it:

-> mvn clean install

Start the WebGoat Server

Run the WebGoat application by executing the following command:

-> java -jar webgoat-server-8.2.3.jar

Ensure you adjust the version number in the command if you have a different version of WebGoat.

Access the Application:

By default, WebGoat runs on port 8080. Open your web browser and navigate to:

-> http://localhost:8080/WebGoat

You should see the WebGoat login page where you can register as a new user or log in with an existing account.

By default, the application runs on port **8080**. Access it via a browser at:

-> http://localhost:8080/WebGoat

Verifying the Installation:

Log in to WebGoat using the default credentials:

Username: Shruthi003
Password: Shruthi@7890

CONFIGURING OWASP ZAP ON KALI LINUX

OWASP ZAP is a critical tool for identifying and exploiting vulnerabilities. The following stepsoutline its configuration on Kali Linux:

Install OWASP ZAP (if not already installed):

OWASP ZAP comes pre-installed in Kali Linux. If missing, install it via:

```
-> sudo apt update
-> sudo apt install zaproxy
```

Launch OWASP ZAP:

From the Terminal:

Open your terminal and type the following command:

```
-> zap
```

From the Applications Menu:

Navigate to the "Applications" menu, search for "OWASP ZAP," and click to launch it.

Update ZAP (Optional but Recommended):

Check for Updates:

Go to "Help" -> "Check for Updates."

If updates are available, follow the on-screen instructions to install them.

Configure Proxy Settings:

Manual Proxy Configuration:

Go to "Tools" -> "Options" -> "Local Proxy."

Set the "Proxy Listen Address" to **127.0.0.1** and the "**Proxy Listen Port"** to **8080** (or any desired port).

Automatic Proxy Configuration (Recommended):

Use the "Dynamically Configure Browsers" feature. This will automatically configure your browser to use ZAP as a proxy.

Start Scanning a Web Application:

- **Site Spider:**
  - Go to the "Sites" tab.
  - Click the "+" button to add a new site.
  - Enter the target URL (e.g., https://www.example.com).
  - Click "Attack" -> "Site Spider."
  - Configure the scan as needed (e.g., attack strength, context, exclusions).
  - Click "Start Scan."

- **Active Scan:**
  - Right-click on a URL in the Sites tab and select "Attack" -> "Active Scan."
  - Configure the scan as needed (e.g., attack strength, context, exclusions).
  - Click "Start Scan."

Setting Up the Testing Environment

To effectively test the WebGoat application, we established a controlled testing environment. This involved:

- **Hardware and Software:** We used a Kali Linux system, a popular choice for security testing, with sufficient RAM and processing power to handle the testing tools and applications.
- **Network Configuration:** To isolate the testing environment and prevent unintended interference, we configured both WebGoat and OWASP ZAP to run locally on the Kali Linux system. This local setup ensured that all traffic between the browser and the application was captured and analyzed by OWASP ZAP.

Application Interaction and Verification

To seamlessly interact with the WebGoat application and verify the correct setup of our testing environment, we took the following steps:

- **Browser Configuration:** We configured Firefox to use OWASP ZAP as a proxy server. This redirected all web traffic from the browser through ZAP, allowing it to intercept and analyze HTTP requests and responses.
- **WebGoat Accessibility:** We ensured that WebGoat was accessible and functional at the specified local address (http://localhost:8080/WebGoat).
- **OWASP ZAP Functionality:** We verified that OWASP ZAP was correctly intercepting traffic and capable of performing vulnerability scans.

Vulnerability Scanning Methodology

We employed a combination of automated and manual testing techniques to identify and assess vulnerabilities in WebGoat:

Automated Scanning with OWASP ZAP

1. **Configuration:** We configured OWASP ZAP to intercept and analyze all traffic between the browser and WebGoat.
2. **Spidering:** We used the "Spider Scan" feature to automatically crawl through WebGoat, discovering all accessible endpoints and input fields.

3. **Active Scanning:** We initiated an Active Scan to systematically test for common web vulnerabilities, including:
    - **SQL Injection (SQLi):** Exploiting vulnerabilities in SQL queries to gain unauthorized access or manipulate data.
    - **Cross-Site Scripting (XSS):** Injecting malicious scripts into web pages to compromise user sessions or steal sensitive information.
    - **Cross-Site Request Forgery (CSRF):** Tricking users into performing unintended actions on a web application.
4. **Reviewing Scan Results:** We carefully examined the vulnerabilities identified by OWASP ZAP, analyzing the request and response data to understand the potential impact and exploitation methods.

Manual Testing

To complement the automated scanning, we performed manual testing to validate and exploit the vulnerabilities identified by OWASP ZAP:

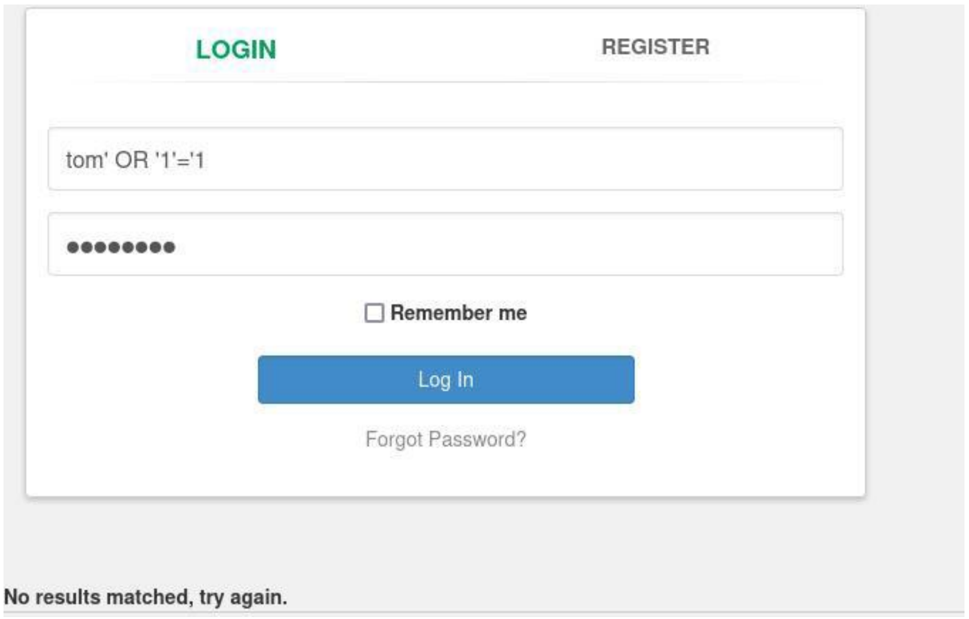
1. **SQL Injection:** We manually tested input fields with carefully crafted SQL injection payloads to bypass authentication mechanisms or extract sensitive data from the database.
2. **Cross-Site Scripting:** We injected malicious JavaScript code into input fields and parameters to test for both reflected and stored XSS vulnerabilities.
3. **Cross-Site Request Forgery:** We crafted malicious HTML forms to mimic legitimate requests, bypassing CSRF protection measures and performing unauthorized actions.

Vulnerability Identification Criteria

To effectively identify and prioritize vulnerabilities, we applied the following criteria:

- **SQL Injection:** We looked for input fields that accepted unsanitized user input and were directly used in SQL queries.
- **Cross-Site Scripting:** We identified input fields that reflected or stored user-provided data without proper sanitization or encoding.
- **Cross-Site Request Forgery:** We assessed the presence of CSRF protection mechanisms, such as synchronization tokens or double-submit cookies.

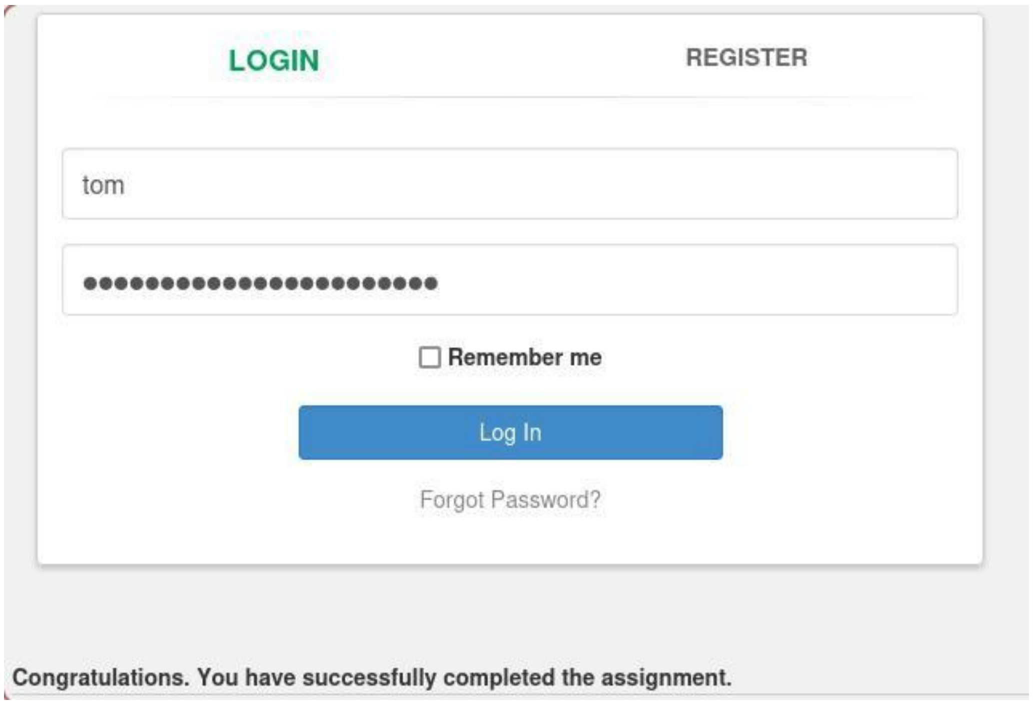
- **Impact and Exploitability:** We evaluated the severity of each vulnerability based on its potential impact (e.g., data breach, unauthorized access) and the ease of exploitation.



tom' OR '1'='1

●●●●●●●●

No results matched, try again.

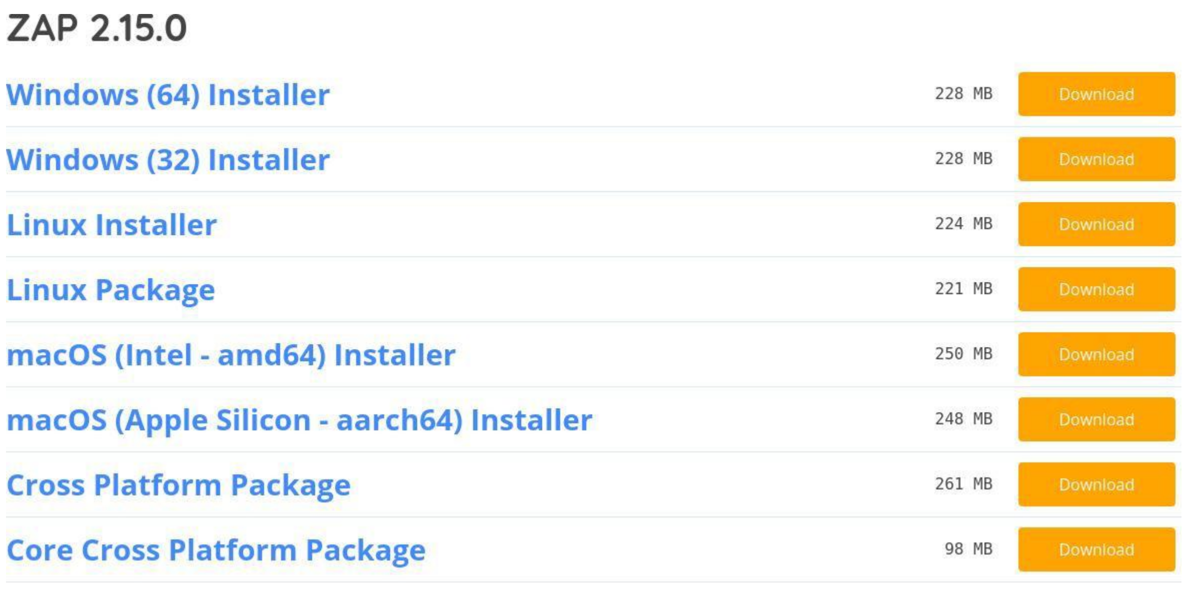


tom

●●●●●●●●●●●●●●●●●●●●●●●●

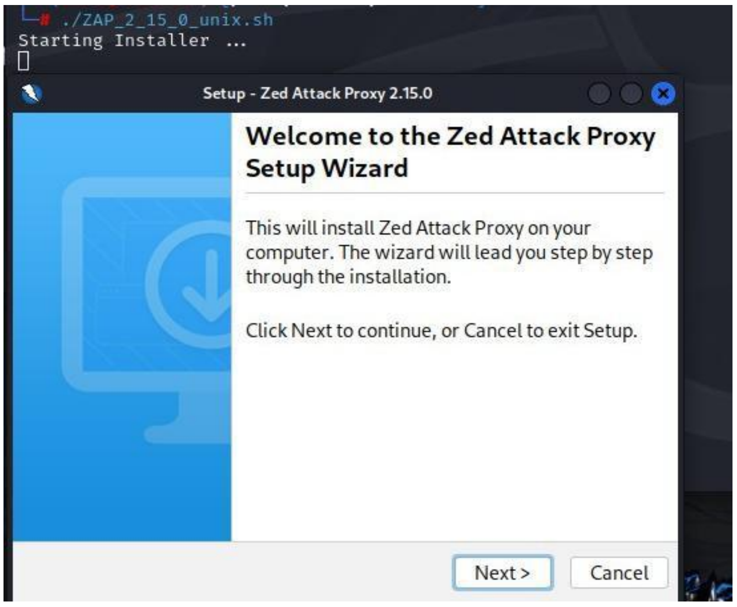
Congratulations. You have successfully completed the assignment.

ZAP INSTALLATION SCREENSHOTS



ZAP 2.15.0

Windows (64) Installer	228 MB	Download
Windows (32) Installer	228 MB	Download
Linux Installer	224 MB	Download
Linux Package	221 MB	Download
macOS (Intel - amd64) Installer	250 MB	Download
macOS (Apple Silicon - aarch64) Installer	248 MB	Download
Cross Platform Package	261 MB	Download
Core Cross Platform Package	98 MB	Download



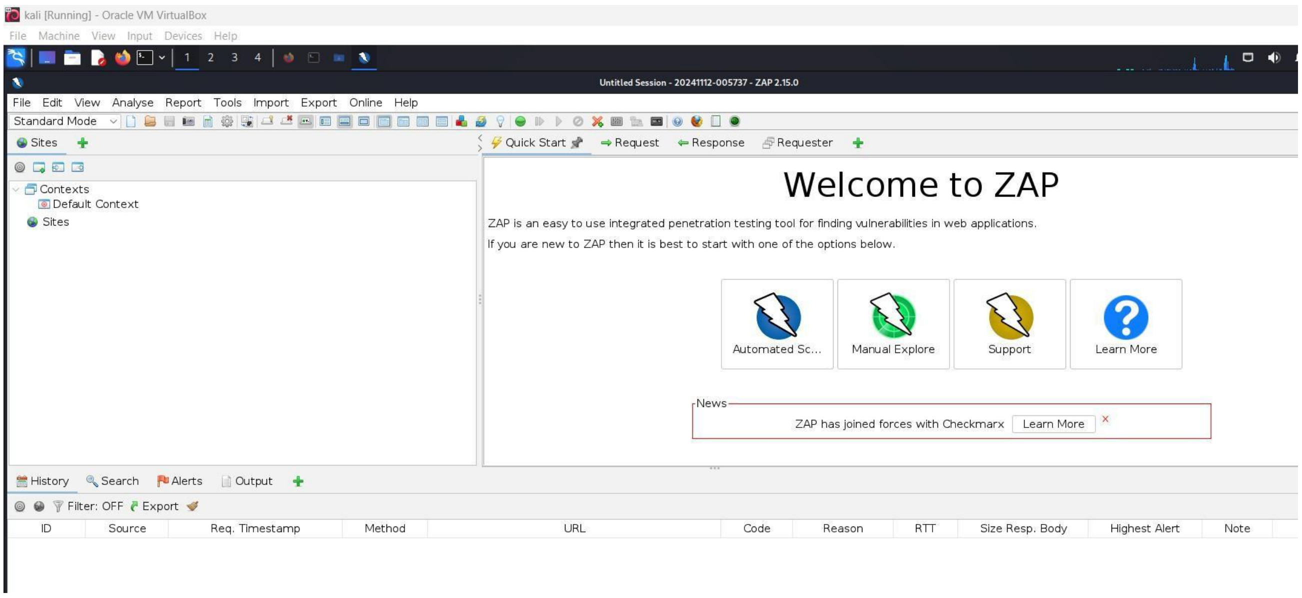
```
└─# ./ZAP_2_15_0_unix.sh
Starting Installer ...
```

Setup - Zed Attack Proxy 2.15.0

Welcome to the Zed Attack Proxy Setup Wizard

This will install Zed Attack Proxy on your computer. The wizard will lead you step by step through the installation.

Click Next to continue, or Cancel to exit Setup.


kali [Running] - Oracle VM VirtualBox

Untitled Session - 20241112-005737 - ZAP 2.15.0

Welcome to ZAP

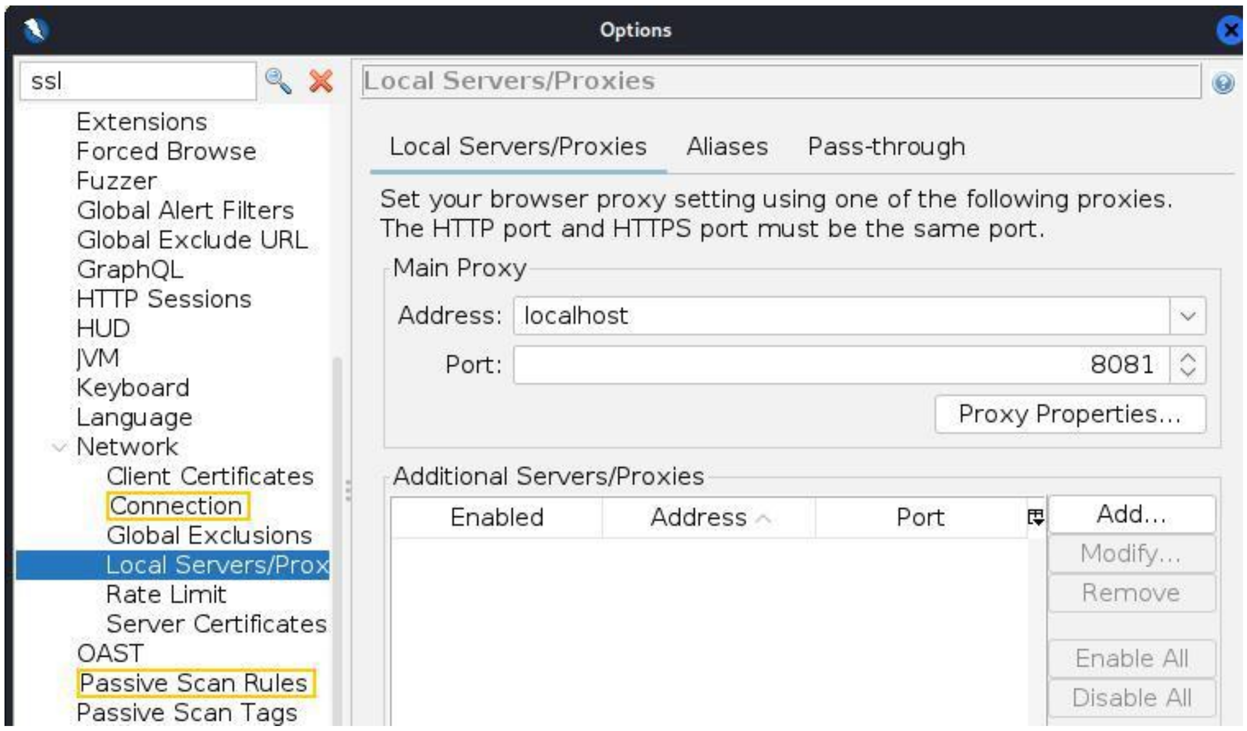
ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

If you are new to ZAP then it is best to start with one of the options below.

News: ZAP has joined forces with Checkmarx

ID	Source	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Body	Highest Alert	Note

Installation done of OWASP ZAP successfully.


Options

Local Servers/Proxies

Set your browser proxy setting using one of the following proxies.
The HTTP port and HTTPS port must be the same port.

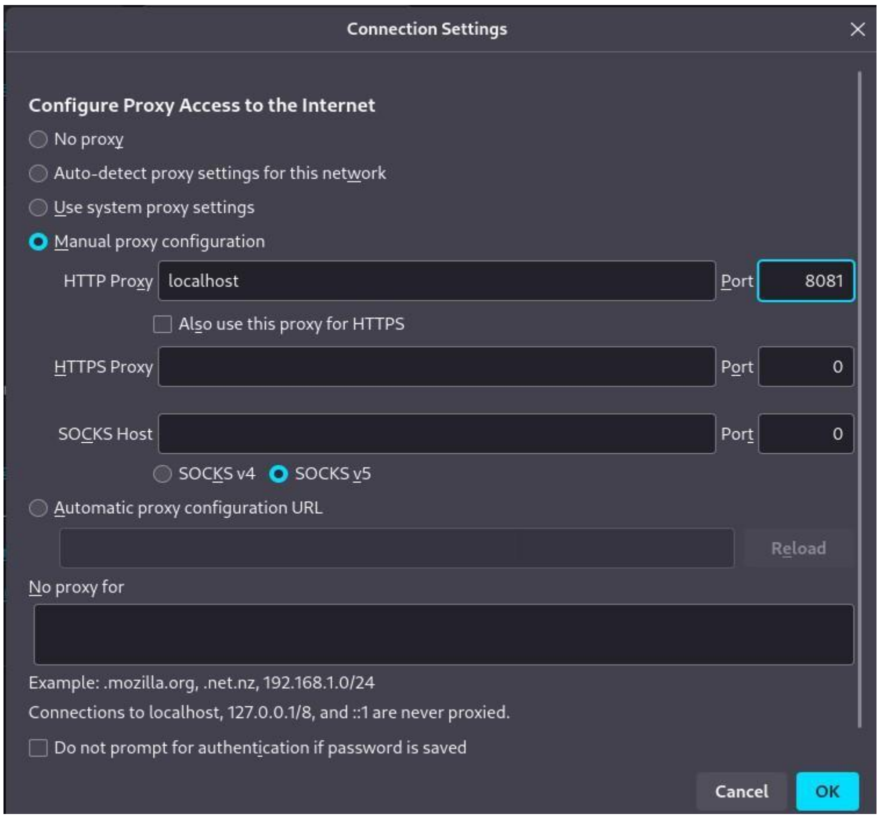
Main Proxy

Address: localhost

Port: 8081

Additional Servers/Proxies

Enabled	Address	Port


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: 8081

☐ Also use this proxy for HTTPS

HTTPS Proxy: Port: 0

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

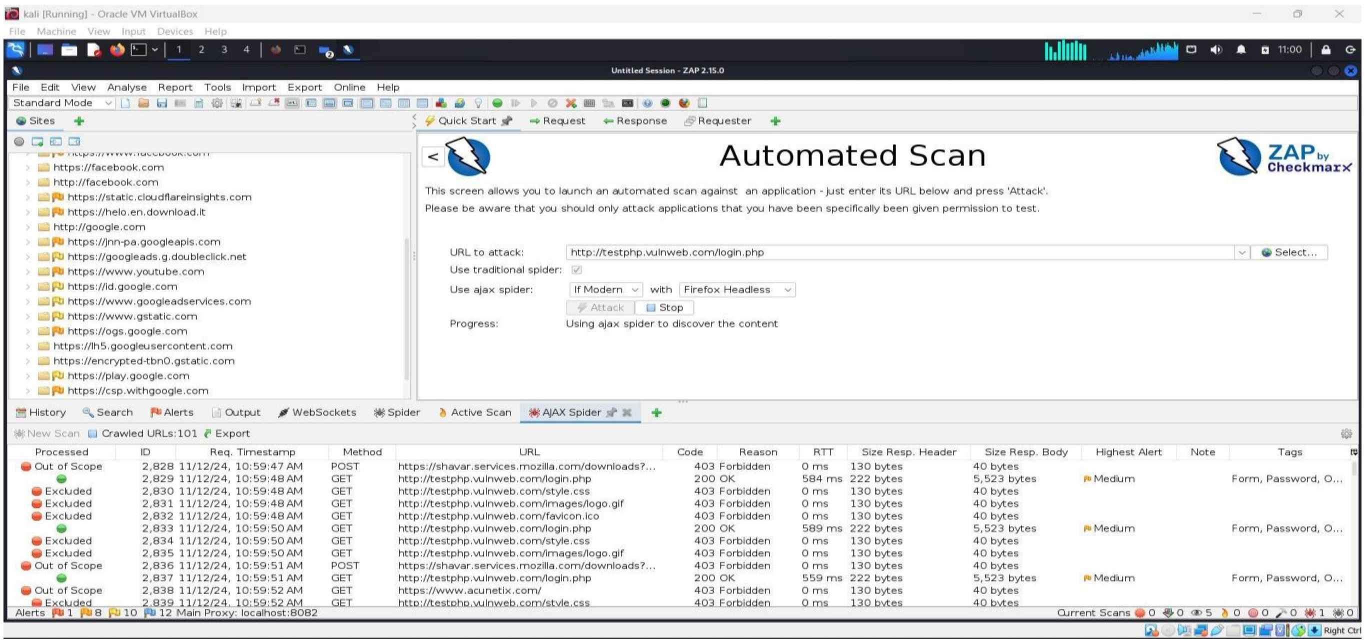
> Generating and installing ZAP SSL certificate

> Export certificate.

> Install certificate in browser.

> For vulnerability analysis, initiate passive scan

> Also run active scan



kali [Running] - Oracle VM VirtualBox

Untitled Session - ZAP 2.15.0

Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.
Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://testphp.vulnweb.com/login.php
Use traditional spider: ✓
Use ajax spider: If Modern with Firefox Headless
Progress: Using ajax spider to discover the content

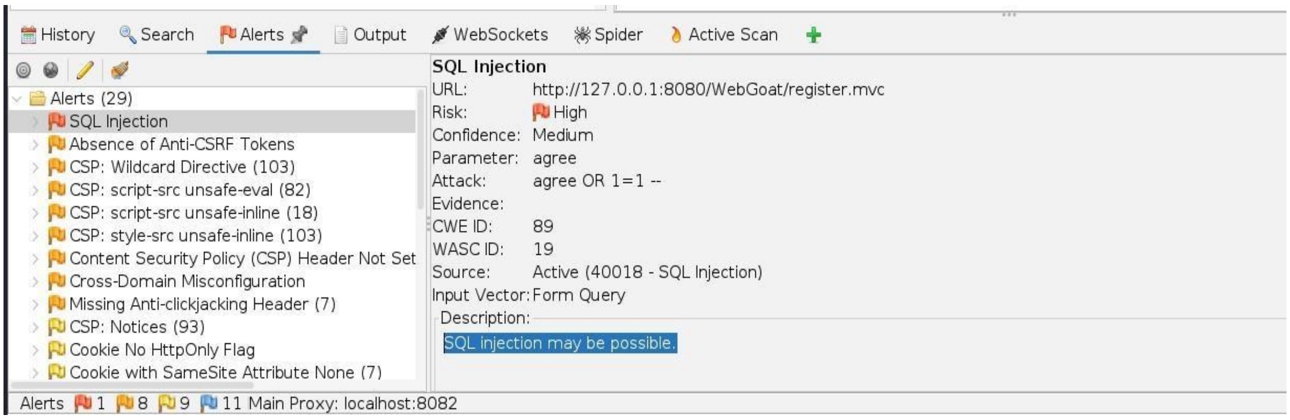
Sites:
- https://facebook.com
- http://facebook.com
- https://static.cloudflareinsights.com
- https://helo.en.download.it
- http://google.com
- https://jnn-pa.googleapis.com
- https://googleads.g.doubleclick.net
- https://www.youtube.com
- https://id.google.com
- https://www.googleadservices.com
- https://www.gstatic.com
- https://ogs.google.com
- https://lh5.googleusercontent.com
- https://encrypted-tbn0.gstatic.com
- https://play.google.com
- https://csp.withgoogle.com

Crawled URLs:101

Processed	ID	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	Note	Tags
Out of Scope	2,828	11/12/24, 10:59:47 AM	POST	https://shavar.services.mozilla.com/downloads?...	403	Forbidden	0 ms	130 bytes	40 bytes			
●	2,829	11/12/24, 10:59:48 AM	GET	http://testphp.vulnweb.com/login.php	200	OK	584 ms	222 bytes	5,523 bytes	Medium		Form, Password, O...
Excluded	2,830	11/12/24, 10:59:48 AM	GET	http://testphp.vulnweb.com/style.css	403	Forbidden	0 ms	130 bytes	40 bytes			
Excluded	2,831	11/12/24, 10:59:48 AM	GET	http://testphp.vulnweb.com/images/logo.gif	403	Forbidden	0 ms	130 bytes	40 bytes			
Excluded	2,832	11/12/24, 10:59:48 AM	GET	http://testphp.vulnweb.com/favicon.ico	403	Forbidden	0 ms	130 bytes	40 bytes			
●	2,833	11/12/24, 10:59:50 AM	GET	http://testphp.vulnweb.com/login.php	200	OK	589 ms	222 bytes	5,523 bytes	Medium		Form, Password, O...
Excluded	2,834	11/12/24, 10:59:50 AM	GET	http://testphp.vulnweb.com/style.css	403	Forbidden	0 ms	130 bytes	40 bytes			
Excluded	2,835	11/12/24, 10:59:50 AM	GET	http://testphp.vulnweb.com/images/logo.gif	403	Forbidden	0 ms	130 bytes	40 bytes			
Out of Scope	2,836	11/12/24, 10:59:51 AM	POST	https://shavar.services.mozilla.com/downloads?...	403	Forbidden	0 ms	130 bytes	40 bytes			
●	2,837	11/12/24, 10:59:51 AM	GET	http://testphp.vulnweb.com/login.php	200	OK	559 ms	222 bytes	5,523 bytes	Medium		Form, Password, O...
Out of Scope	2,838	11/12/24, 10:59:52 AM	GET	https://www.acunetix.com/	403	Forbidden	0 ms	130 bytes	40 bytes			
Excluded	2,839	11/12/24, 10:59:52 AM	GET	http://testphp.vulnweb.com/style.css	403	Forbidden	0 ms	130 bytes	40 bytes			

Alerts 1 8 10 12 Main Proxy: localhost:8082 Current Scans 0 0 5 0 0 0 1 0

. > Vulnerabilities finding



Alerts (29)
- SQL Injection
- Absence of Anti-CSRF Tokens
- CSP: Wildcard Directive (103)
- CSP: script-src unsafe-eval (82)
- CSP: script-src unsafe-inline (18)
- CSP: style-src unsafe-inline (103)
- Content Security Policy (CSP) Header Not Set
- Cross-Domain Misconfiguration
- Missing Anti-clickjacking Header (7)
- CSP: Notices (93)
- Cookie No HttpOnly Flag
- Cookie with SameSite Attribute None (7)

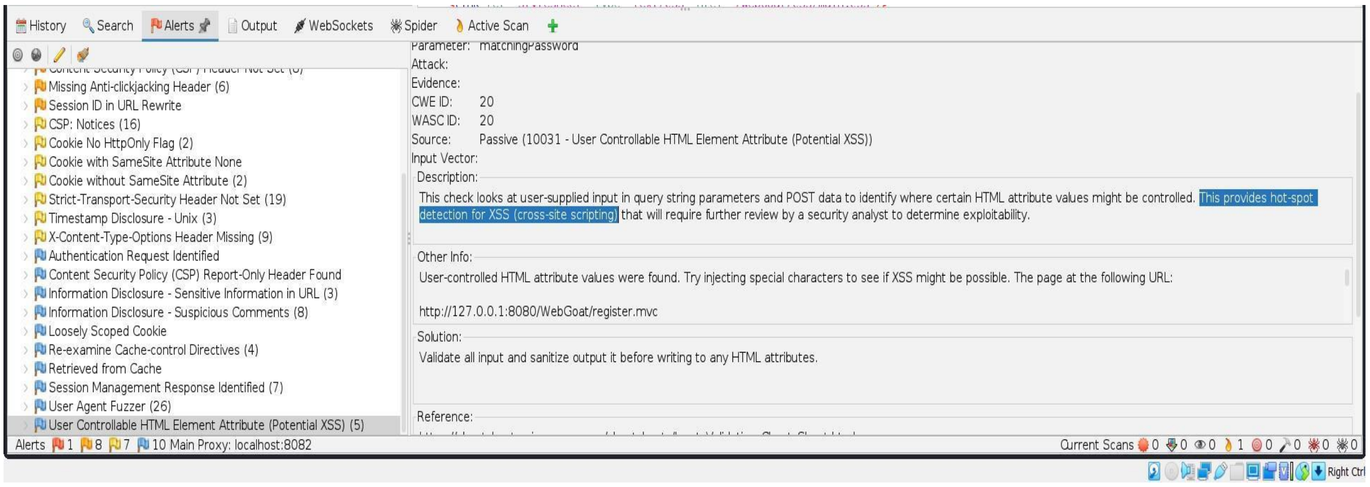
SQL Injection

URL:	http://127.0.0.1:8080/WebGoat/register.mvc
Risk:	High
Confidence:	Medium
Parameter:	agree
Attack:	agree OR 1=1 --
Evidence:	
CWE ID:	89
WASC ID:	19
Source:	Active (40018 - SQL Injection)
Input Vector:	Form Query

Description:

SQL injection may be possible.

Alerts 1 8 9 11 Main Proxy: localhost:8082



History | Search | Alerts | Output | WebSockets | Spider | Active Scan

Missing Anti-clickjacking Header (6)
Session ID in URL Rewrite
CSP: Notices (16)
Cookie No HttpOnly Flag (2)
Cookie with SameSite Attribute None
Cookie without SameSite Attribute (2)
Strict-Transport-Security Header Not Set (19)
Timestamp Disclosure - Unix (3)
X-Content-Type-Options Header Missing (9)
Authentication Request Identified
Content Security Policy (CSP) Report-Only Header Found
Information Disclosure - Sensitive Information in URL (3)
Information Disclosure - Suspicious Comments (8)
Loosely Scoped Cookie
Re-examine Cache-control Directives (4)
Retrieved from Cache
Session Management Response Identified (7)
User Agent Fuzzer (26)
User Controllable HTML Element Attribute (Potential XSS) (5)

```
Parameter:  matchingPassword
Attack:
Evidence:
CWE ID:     20
WASC ID:    20
Source:     Passive (10031 - User Controllable HTML Element Attribute (Potential XSS))
Input Vector:
```

Description:
This check looks at user-supplied input in query string parameters and POST data to identify where certain HTML attribute values might be controlled. This provides hot-spot detection for XSS (cross-site scripting) that will require further review by a security analyst to determine exploitability.

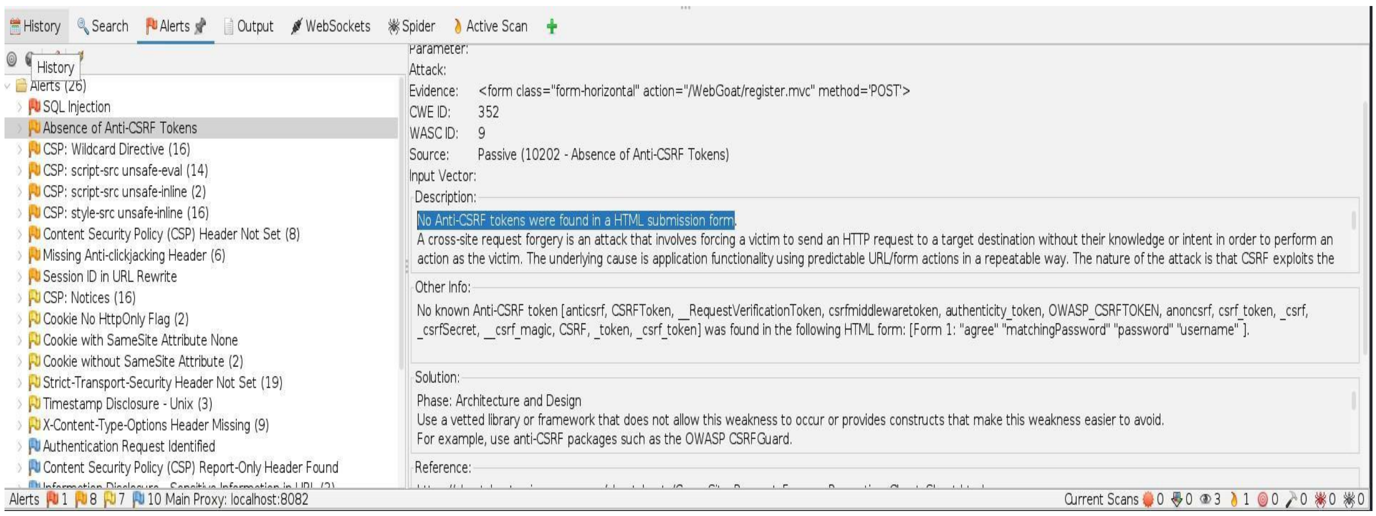
Other Info:
User-controlled HTML attribute values were found. Try injecting special characters to see if XSS might be possible. The page at the following URL:

http://127.0.0.1:8080/WebGoat/register.mvc

Solution:
Validate all input and sanitize output it before writing to any HTML attributes.

Reference:

Alerts 1 8 7 10 Main Proxy: localhost:8082 — Current Scans 0 0 0 1 0 0 0 0



History | Search | Alerts | Output | WebSockets | Spider | Active Scan

Alerts (26)
SQL Injection
Absence of Anti-CSRF Tokens
CSP: Wildcard Directive (16)
CSP: script-src unsafe-eval (14)
CSP: script-src unsafe-inline (2)
CSP: style-src unsafe-inline (16)
Content Security Policy (CSP) Header Not Set (8)
Missing Anti-clickjacking Header (6)
Session ID in URL Rewrite
CSP: Notices (16)
Cookie No HttpOnly Flag (2)
Cookie with SameSite Attribute None
Cookie without SameSite Attribute (2)
Strict-Transport-Security Header Not Set (19)
Timestamp Disclosure - Unix (3)
X-Content-Type-Options Header Missing (9)
Authentication Request Identified
Content Security Policy (CSP) Report-Only Header Found

```
Parameter:
Attack:
Evidence:   <form class="form-horizontal" action="/WebGoat/register.mvc" method='POST'>
CWE ID:     352
WASC ID:    9
Source:     Passive (10202 - Absence of Anti-CSRF Tokens)
Input Vector:
```

Description:
No Anti-CSRF tokens were found in a HTML submission form.
A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the

Other Info:
No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, _csrfSecret, __csrf_magic, CSRF, _token, _csrf_token] was found in the following HTML form: [Form 1: "agree" "matchingPassword" "password" "username"].

Solution:
Phase: Architecture and Design
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
For example, use anti-CSRF packages such as the OWASP CSRFGuard.

Reference:

Alerts 1 8 7 10 Main Proxy: localhost:8082 — Current Scans 0 0 3 1 0 0 0 0

XSS Exploitation

- Vulnerability Location

- **Location:** The vulnerability was identified in the comments section of the WebGoat application.
- **Affected Parameter:** The specific input field that caused the issue was the comment input field. This field allowed users to input text, which was then reflected back to the user without proper sanitization.

- Exploitation Process

Discovery:

- **Automated Scan:** OWASP ZAP, a web application security scanner, detected a potential reflected XSS vulnerability in the comments section during its active scan.
- **Unsanitized Input:** The scanner identified that user-provided input in the comment field was being directly reflected back to the user without any sanitization or encoding. This lack of filtering allowed malicious scripts to be executed within the context of the vulnerable web page.

Exploit:

> **Malicious Payload:** The following XSS payload was crafted:

```
HTML
<script>alert('XSS Exploited');</script>
```

> **Payload Injection:** This payload was submitted through the comment input field.

> **Script Execution:** When the comment was posted and the page reloaded, the malicious script was executed within the user's browser, resulting in an alert box displaying the message "XSS Exploited."

This successful exploitation demonstrated the severity of the XSS vulnerability. By injecting malicious scripts, attackers could potentially steal sensitive information, hijack user sessions, or redirect users to malicious websites.

HOW TO EXPLOIT - PROCEDURE

Discovery:

- **OWASP ZAP Scan:** During the active scan, OWASP ZAP identified a vulnerability in the password reset form of the WebGoat application.
- **Lack of CSRF Protection:** The scanner found that the password reset request relied solely on user authentication cookies for protection. This meant that an attacker could potentially craft malicious requests and trick an authenticated user into submitting them, leading to unauthorized password changes.

Exploit:

Malicious Form: The attacker creates a malicious HTML form that targets the password reset endpoint of the WebGoat application.

HTML

```html
<form action="http://localhost:8080/WebGoat/reset-password" method="POST">
   <input type="hidden" name="password" value="maliciouspassword123">
   <input type="hidden" name="confirmPassword" value="maliciouspassword123">
   <input type="submit" value="Submit"></form>
```

- **Tricking the Victim:** The attacker tricks a legitimate, authenticated user into clicking on this malicious form.
- **Unauthorized Password Reset:** When the victim clicks the submit button, the browser sends a POST request to the WebGoat server, including the malicious password values. Since the request originates from the victim's browser and contains their valid authentication cookie, the server processes the request as if it came directly from the victim.
- **Successful Password Change:** As a result, the victim's password is changed to the malicious password without their knowledge or consent.

This successful CSRF attack highlights the importance of implementing strong CSRF protection mechanisms to prevent unauthorized actions on behalf of authenticated users.

Vulnerability Mitigation Strategies

Securing Against SQL Injection

SQL Injection attacks occur when malicious input is inserted into SQL queries, potentially leading to unauthorized data access, modification, or deletion. To mitigate this risk, consider the following strategies:

- Parameterized Queries:

Use parameterized queries or prepared statements to separate SQL code from user-provided data. This prevents SQL injection by treating user input as data, not as part of the query.

- Input Validation:

Implement strict input validation to filter out malicious characters and enforce data type constraints. This helps prevent the injection of harmful SQL commands.

- Least Privilege Principle:

Grant database users only the minimum necessary permissions to perform their tasks. This limits the potential damage if an attacker gains unauthorized access.

-Error Handling:

Avoid revealing sensitive error messages that could provide clues to attackers. Instead, use generic error messages and log detailed error information for analysis.

- Regular Security Audits:

Conduct regular security audits to identify and address potential vulnerabilities.

Securing Against Cross-Site Scripting (XSS)

XSS attacks allow attackers to inject malicious scripts into web pages, which can be executed by unsuspecting users. To mitigate this risk, consider the following strategies:

- Input Validation and Sanitization:

Thoroughly validate and sanitize all user input to remove or encode malicious characters.

- Output Encoding:

Properly encode output to prevent the execution of malicious scripts. This includes HTML encoding, JavaScript encoding, and CSS encoding.

- Content Security Policy (CSP):

Implement a CSP to restrict the sources of scripts, stylesheets, and other resources that can be loaded by the browser.

- HTTP-Only Cookies:

Use HTTP-Only cookies to prevent client-side scripts from accessing sensitive session cookies.

- Secure Headers:

Set appropriate security headers, such as X-Frame-Options, X-XSS-Protection, and Strict-Transport-Security, to enhance security.

Securing Against Cross-Site Request Forgery (CSRF)

CSRF attacks exploit the trust between a user and a web application to trick the user into performing unintended actions. To mitigate this risk, consider the following strategies:

- CSRF Tokens:

Generate unique, session-specific tokens and include them in both the form and the request. Validate these tokens on the server-side to ensure the request is legitimate.

- Referer Header Checking:

Check the Referer header to verify that the request originated from a trusted source.

- SameSite Cookies:

Use the SameSite attribute in cookies to prevent them from being sent in cross-site requests.

Conclusion

Summary of Findings:

This project successfully identified, exploited, and documented common web application vulnerabilities, including SQL Injection, XSS, and CSRF. By understanding these vulnerabilities and implementing effective mitigation strategies, organizations can significantly enhance the security of their web applications.

Importance of Remediation:

Addressing these vulnerabilities is crucial to protect sensitive data, maintain user trust, and comply with security regulations. Failure to do so can lead to serious consequences, such as data breaches, financial loss, and reputational damage.

Future Work and Recommendations:

To further strengthen web application security, consider the following recommendations:

- **Automated Security Testing:** Integrate automated security testing tools into the development and deployment pipelines to identify vulnerabilities early in the development process.
- **Secure Coding Practices:** Promote secure coding practices among developers, such as input validation, output encoding, and proper error handling.
- **Web Application Firewalls (WAFs):** Deploy WAFs to provide an additional layer of protection against web attacks.
- **Regular Security Audits and Penetration Testing:** Conduct regular security assessments to identify and address new vulnerabilities.
- **Stay Informed:** Keep up-to-date with the latest security trends and best practices.

By following these recommendations and continuously improving security measures, organizations can significantly reduce the risk of web application attacks.