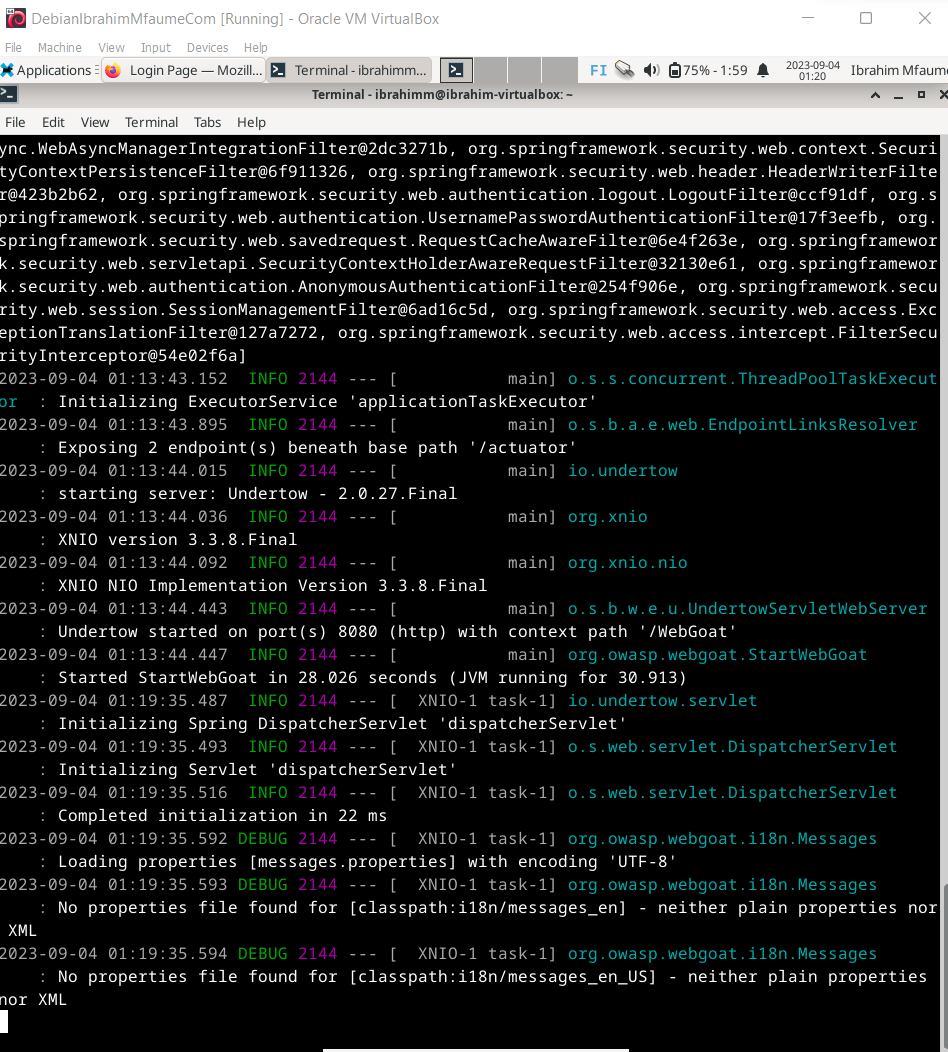
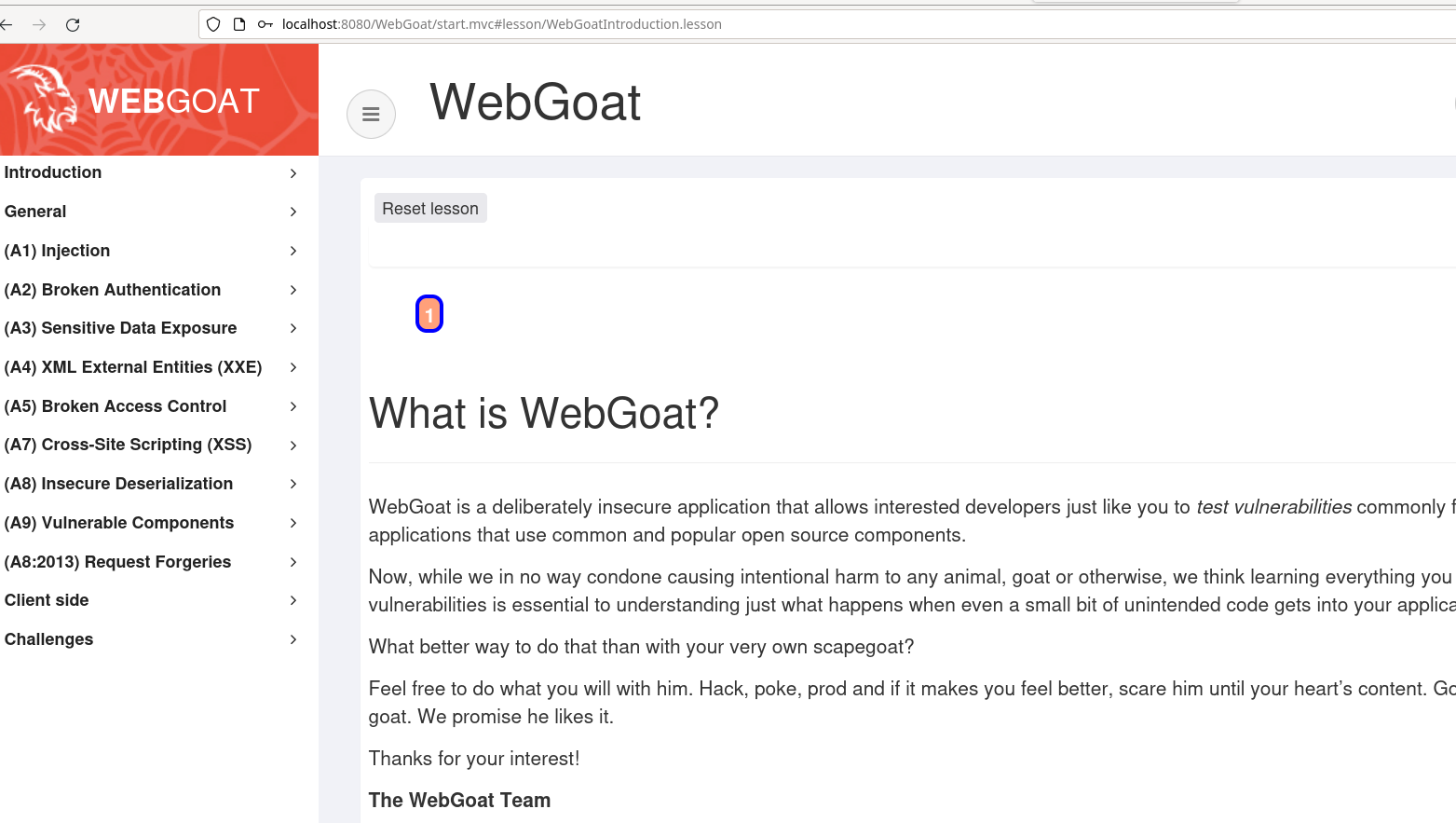
1. **install webgoat**

installed successful!

****

User registration done

****

**x) OWASP: OWASP 10 2021**

* [**A05:2021-Security Misconfiguration**](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)

The application may be vulnerable if it lacks proper security measures, includes unnecessary features, contains default account settings, exposes sensitive error information, disables security on upgrades, has insufficient security settings, lacks security headers, or uses outdated software.

**Risk Prevention**

To prevent vulnerabilities, ensure secure installations with a repeatable hardening process, regular configuration updates, client security updates, and automated configuration verification across all environments etc.

**Example of an attack scenario**

A security issue where sample applications with known vulnerabilities are left on the production server, including an admin console with unchanged default login credentials. This mistake allows an attacker to easily gain access the admin console and compromise the server by using these known flaws.

**A06:2021 – Vulnerable and Outdated Components**

**You are vulnerable if:**

1. You're unaware of component versions, both client-side and server-side, including nested dependencies.

2. Your software is outdated, unsupported, or vulnerable, including OS, servers, applications, and libraries.

3. You don't regularly scan for vulnerabilities or subscribe to security bulletins.

4. You delay fixing or upgrading your platform, frameworks, and dependencies.

5. Developers don't test library compatibility.

6. You neglect securing component configurations.

To prevent vulnerabilities:

**To prevent**

1. Remove unused components.

2. Track versions and vulnerabilities.

3. Use official sources for components.

4. Monitor and address unpatched libraries.

5. Maintain an ongoing update plan.

Scenario, component vulnerabilities can have a serious impact on the application's security, whether they result from coding errors or intentional backdoors.

# A03:2021 – Injection

Injection vulnerabilities ranked third in the study, with 94% of applications tested showing some form of risk. Common weaknesses included Cross-site Scripting, SQL Injection, and External Control of File Name or Path. These vulnerabilities occurred frequently, totaling 274,000 instances.

An application is vulnerable to attack when it mishandles user-supplied data, uses dynamic queries without proper security measures, misuses data in object-relational mapping, or incorporates hostile data directly in SQL or command structures.

To prevent injection attacks:

1. Separate data from commands and queries:

- Prefer using a safe API, parameterized interfaces, or Object Relational Mapping Tools (ORMs).

- Be cautious with stored procedures that concatenate queries and data.

2. Implement server-side input validation, but remember that it's not a complete defense.

3. For remaining dynamic queries, escape special characters using the interpreter's specific escape syntax.

4. Employ SQL controls like LIMIT to prevent mass record disclosure in case of SQL injection.

* d) Sequel. Solve [SQLZoo](https://sqlzoo.net/wiki/SQL_Tutorial):
  + 0 SELECT basics

A screenshot of a computer

Description automatically generated

* + 2 SELECT from World, from first two subtasks.

A screenshot of a computer

Description automatically generated

* **e) Johnny tables. Solve Portswigger Labs:**[**Lab: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data**](https://portswigger.net/web-security/sql-injection/lab-retrieve-hidden-data)

