



Veermata Jijabai Technological Institute, Mumbai 400019

Experiment No.: 07

Aim: Consider the systems software's and applications software are loaded on web server, application server and database server. Gather the information about the enterprise for making the attacks. And Perform any 4 Top 10 OWASP attack preferably.

- a) Broken Authentication Attack(Identification and Authentication Failure)
- b) Any injection attack
- c) Security Misconfiguration attack
- d) Broken Access Control

Group : Kiran Patil - 211070904

Mayuresh Murudkar - 211070903

Pratiksha Sankhe – 201071049

Branch: Final Year B.Tech Computer Engineering

Batch: D

OWASP Attacks

THEORY

OWASP

The Open Web Application Security Project, or OWASP, is an international non-profit organization dedicated to web application security. One of OWASP's core principles is that all of their materials be freely available and easily accessible on their website, making it possible for anyone to improve their own web application security. The materials they offer include documentation, tools, videos, and forums. Perhaps their best-known project is the OWASP Top 10.

What is the OWASP Top 10?

The OWASP Top 10 is a regularly-updated report outlining security concerns for web application security, focusing on the 10 most critical risks. The report is put together by a team of security experts from all over the world. OWASP refers to the Top 10 as an 'awareness document' and they recommend that all companies incorporate the report into their processes in order to minimize and/or mitigate security risks.

Below are the security risks reported in the OWASP Top 10 2017 report:

1. Injection

Injection attacks happen when untrusted data is sent to a code interpreter through a form input or some other data submission to a web application. For example, an attacker could enter SQL database code into a form that expects a plaintext username. If that form input is not properly secured, this would result in that SQL code being executed. This is known as an SQL injection attack. Injection attacks can be prevented by validating and/or sanitizing user-submitted data. (Validation means rejecting suspicious-looking data, while sanitization refers to cleaning up the suspicious-looking parts of the data.) In addition, a database admin can set controls to minimize the amount of information an injection attack can expose.

2. Broken Authentication

Vulnerabilities in authentication (login) systems can give attackers access to user accounts and even the ability to compromise an entire system using an admin account. For example, an attacker can take a list containing thousands of known username/password combinations obtained during a data breach and use a script to try all those combinations on a login system to see if there are any that work. Some strategies to mitigate authentication vulnerabilities are requiring two-factor authentication (2FA) as well as limiting or delaying repeated login attempts using rate limiting.

3. Sensitive Data Exposure

If web applications don't protect sensitive data such as financial information and passwords, attackers can gain access to that data and sell or utilize it for nefarious purposes. One popular method for stealing sensitive information is using an on-path attack. Data exposure risk can be minimized by encrypting all sensitive data as well as disabling the caching* of any sensitive information. Additionally, web application developers should take care to ensure that they are not unnecessarily storing any sensitive data. Caching is the practice of temporarily storing data for re-use. For example, web browsers will often cache webpages so that if a user revisits those pages within a fixed time span, the browser does not have to fetch the pages from the web.

4. XML External Entities (XEE)

This is an attack against a web application that parses XML* input. This input can reference an external entity, attempting to exploit a vulnerability in the parser. An 'external entity' in this context refers to a storage unit, such as a hard drive. An XML parser can be duped into sending data to an unauthorized external entity, which can pass sensitive data directly to an attacker. The best ways to prevent XEE attacks are to have web applications accept a less complex type of data, such as JSON**, or at the very least to patch XML parsers and disable the use of external entities in an XML application. XML or Extensible Markup Language is a markup language intended to be both human-readable and machine-readable. Due to its complexity and security vulnerabilities, it is now being phased out of use in many web applications. JavaScript Object Notation (JSON) is a type of simple, human-readable notation often used to transmit data over the internet. Although it was originally created for JavaScript, JSON is language-agnostic and can be interpreted by many different programming languages.

5. Broken Access Control

Access control refers to a system that controls access to information or functionality. Broken access controls allow attackers to bypass authorization and perform tasks as though they were privileged users such as administrators. For example a web application could allow a user to change which account they are logged in as simply by changing part of a url, without any other verification. Access controls can be secured by ensuring that a web application uses authorization tokens* and sets tight controls on them. Many services issue authorization tokens when users log in. Every privileged request that a user makes will require that the authorization token be present. This is a secure way to ensure that the user is who they say they are, without having to constantly enter their login credentials.

6. Security Misconfiguration

Security misconfiguration is the most common vulnerability on the list, and is often the result of using default configurations or displaying excessively verbose errors. For instance, an application could show a user overly-descriptive errors which may reveal vulnerabilities in the application. This can be mitigated by removing any unused features in the code and ensuring that error messages are more general.

7. Cross-Site Scripting

Cross-site scripting vulnerabilities occur when web applications allow users to add custom code into a url path or onto a website that will be seen by other users. This vulnerability can be exploited to run malicious JavaScript code on a victim's browser. For example, an attacker could send an email to a victim that appears to be from a trusted bank, with a link to that bank's website. This link could have some malicious JavaScript code tagged onto the end of the url. If the bank's site is not properly protected against cross-site scripting, then that malicious code will be run in the victim's web browser when they click on the link. Mitigation strategies for cross-site scripting include escaping untrusted HTTP requests as well as validating and/or sanitizing user-generated content. Using modern web development frameworks like ReactJS and Ruby on Rails also provides some built-in cross-site scripting protection.

8. Insecure Deserialization

This threat targets the many web applications which frequently serialize and deserialize data. Serialization means taking objects from the application code and converting them into a format that can be used for another purpose, such as storing the data to disk or streaming it. Deserialization is just the opposite: converting serialized data back into objects the application can use. Serialization is sort of like packing furniture away into boxes before a move, and deserialization is like unpacking the boxes and assembling the furniture after the move. An insecure deserialization attack is like having the movers tamper with the contents of the boxes before they are unpacked. An insecure deserialization exploit is the result of deserializing data from untrusted sources, and can result in serious consequences like DDoS attacks and remote code execution attacks. While steps can be taken to try and catch attackers, such as monitoring deserialization and implementing type checks, the only sure way to protect against insecure deserialization attacks is to prohibit the deserialization of data from untrusted sources.

9. Using Components with Known Vulnerabilities

Many modern web developers use components such as libraries and frameworks in their web applications. These components are pieces of software that help developers avoid redundant work and provide needed functionality; common example include front-end frameworks like React and smaller libraries that used to add share icons or a/b testing. Some attackers look for

vulnerabilities in these components which they can then use to orchestrate attacks. Some of the more popular components are used on hundreds of thousands of websites; an attacker finding a security hole in one of these components could leave hundreds of thousands of sites vulnerable to exploit. Component developers often offer security patches and updates to plug up known vulnerabilities, but web application developers don't always have the patched or most-recent versions of components running on their applications. To minimize the risk of running components with known vulnerabilities, developers should remove unused components from their projects, as well as ensuring that they are receiving components from a trusted source and ensuring they are up to date.

10. Insufficient Logging And Monitoring

Many web applications are not taking enough steps to detect data breaches. The average discovery time for a breach is around 200 days after it has happened. This gives attackers a lot of time to cause damage before there is any response. OWASP recommends that web developers should implement logging and monitoring as well as incident response plans to ensure that they are made aware of attacks on their applications.

Broken Authentication Attack

Steps to attack

1. Check if java is installed on your system if not then install it

```
pratiksha@pratiksha-ubuntu:~$ java - version

Command 'java' not found, but can be installed with:

sudo apt install openjdk-11-jre-headless # version 11.0.19+7~us1-0ubuntu1~20.04.1, or
sudo apt install default-jre            # version 2:1.11-72
sudo apt install openjdk-13-jre-headless # version 13.0.7+5-0ubuntu1~20.04
sudo apt install openjdk-16-jre-headless # version 16.0.1+9-1~20.04
sudo apt install openjdk-17-jre-headless # version 17.0.7+7~us1-0ubuntu1~20.04
sudo apt install openjdk-8-jre-headless  # version 8u372-ga~us1-0ubuntu1~20.04
```

```
pratiksha@pratiksha-ubuntu:~$ sudo apt install default-jdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ca-certificates-java default-jdk-headless default-jre default-jre-headless
  fonts-dejavu-extra java-common libatk-wrapper-java libatk-wrapper-java-jni
  openjdk-11-jdk openjdk-11-jdk-headless openjdk-11-jre
  openjdk-11-jre-headless
Suggested packages:
  openjdk-11-demo openjdk-11-source visualvm fonts-ipafont-gothic
  fonts-ipafont-mincho fonts-wqy-microhei | fonts-wqy-zenhei
The following NEW packages will be installed:
  ca-certificates-java default-jdk default-jdk-headless default-jre
  default-jre-headless fonts-dejavu-extra java-common libatk-wrapper-java
  libatk-wrapper-java-jni openjdk-11-jdk openjdk-11-jdk-headless
  openjdk-11-jre openjdk-11-jre-headless
0 upgraded, 13 newly installed, 0 to remove and 264 not upgraded.
Need to get 116 MB of archives.
After this operation, 268 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://in.archive.ubuntu.com/ubuntu focal/main amd64 java-common all 0.72
[6,816 B]
Get:2 http://in.archive.ubuntu.com/ubuntu focal-updates/main amd64 openjdk-11-j
re-headless amd64 11.0.21+9-0ubuntu1~20.04 [38.3 MB]
```

2. Install webgoat

```
pratiksha@pratiksha-ubuntu:~$ wget https://github.com/WebGoat/WebGoat/releases/download/v2023.5/webgoat-2023.5.jar
--2023-11-30 21:56:53-- https://github.com/WebGoat/WebGoat/releases/download/v2023.5/webgoat-2023.5.jar
Resolving github.com (github.com)... 20.207.73.82
Connecting to github.com (github.com)|20.207.73.82|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/31771754/1a58d377-3c2c-46df-b1b3-9956a389e3f6?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20231130%2Fus-east-1%2Fus-east-1%2Faws4_request&X-Amz-Date=20231130T162654Z&X-Amz-Expires=300&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=31771754&response-content-disposition=attachment%3B%20filename%3Dwebgoat-2023.5.jar&response-content-type=application%2Foctet-stream [following]
--2023-11-30 21:56:54-- https://objects.githubusercontent.com/github-production-release-asset-2e65be/31771754/1a58d377-3c2c-46df-b1b3-9956a389e3f6?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20231130%2Fus-east-1%2Fus-east-1%2Faws4_request&X-Amz-Date=20231130T162654Z&X-Amz-Expires=300&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=31771754&response-content-disposition=attachment%3B%20filename%3Dwebgoat-2023.5.jar&response-content-type=application%2Foctet-stream
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 118104362 (113M) [application/octet-stream]
Saving to: 'webgoat-2023.5.jar'

webgoat-2023.5.jar  14%[====>] 16.47M  389KB/s  eta 3m 43s
```

3. Start the burpsuite browser

← → ↻ ⓘ localhost:8080/WebGoat/register.mvc

WEBGOAT

Register

Username size must be between 6 and 45

Password size must be between 6 and 10

Confirm password size must be between 6 and 10

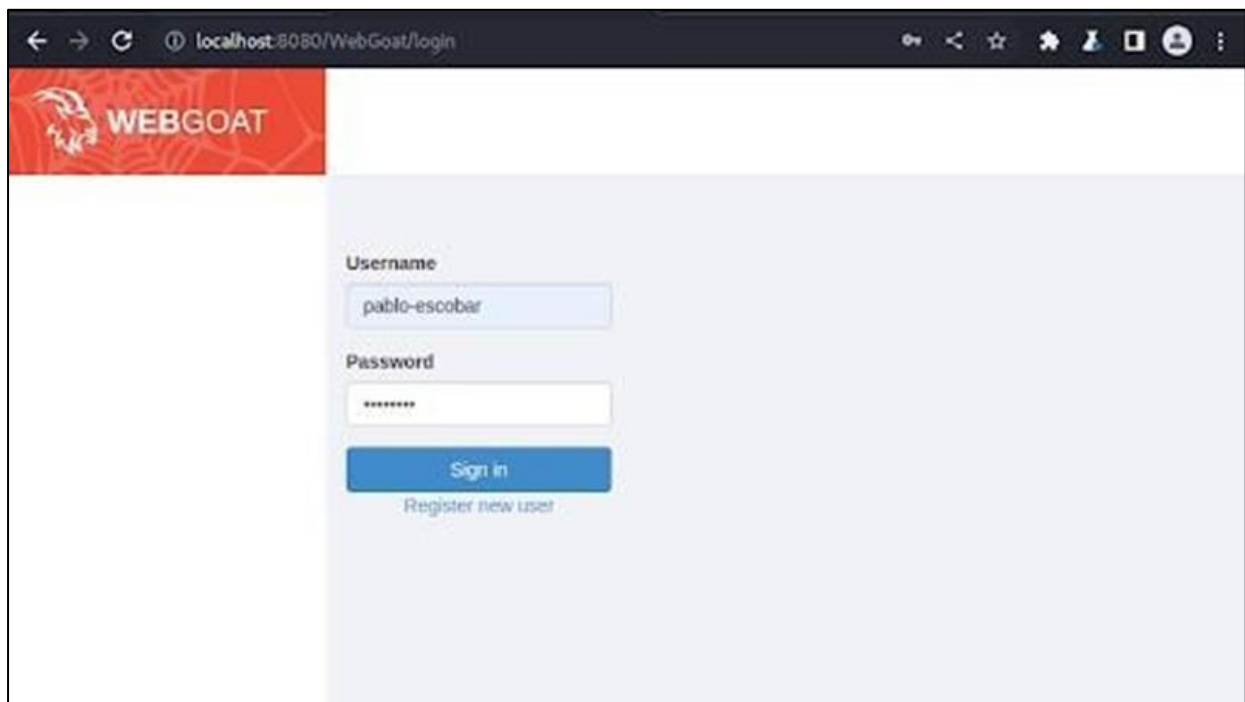
Terms of use

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.

This program is for educational purposes only. If you attempt these techniques without authorization, you are very likely to get

☒ Agree with the terms and conditions

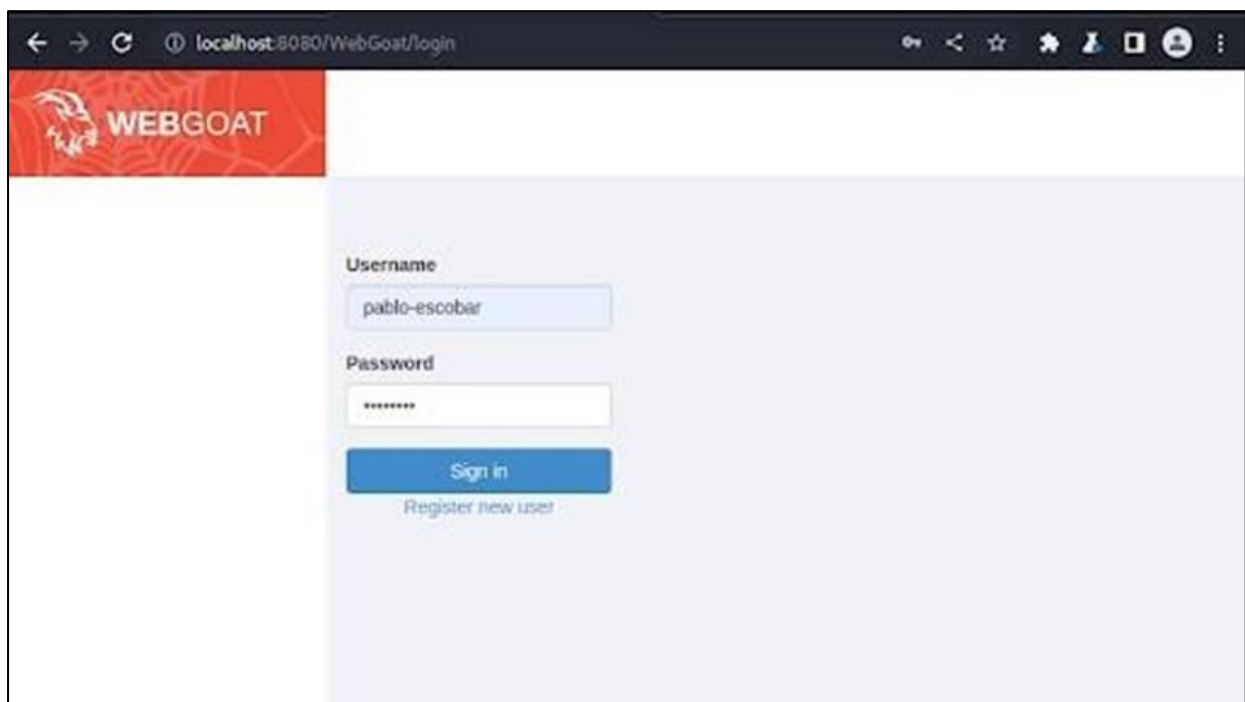
4. Go to register page of webgoat and then create a new profile



A screenshot of a web browser window showing the WebGoat login page. The browser's address bar displays 'localhost:8080/WebGoat/login'. The page features a red header with the 'WEBGOAT' logo. The main content area is light blue and contains a login form with the following elements:

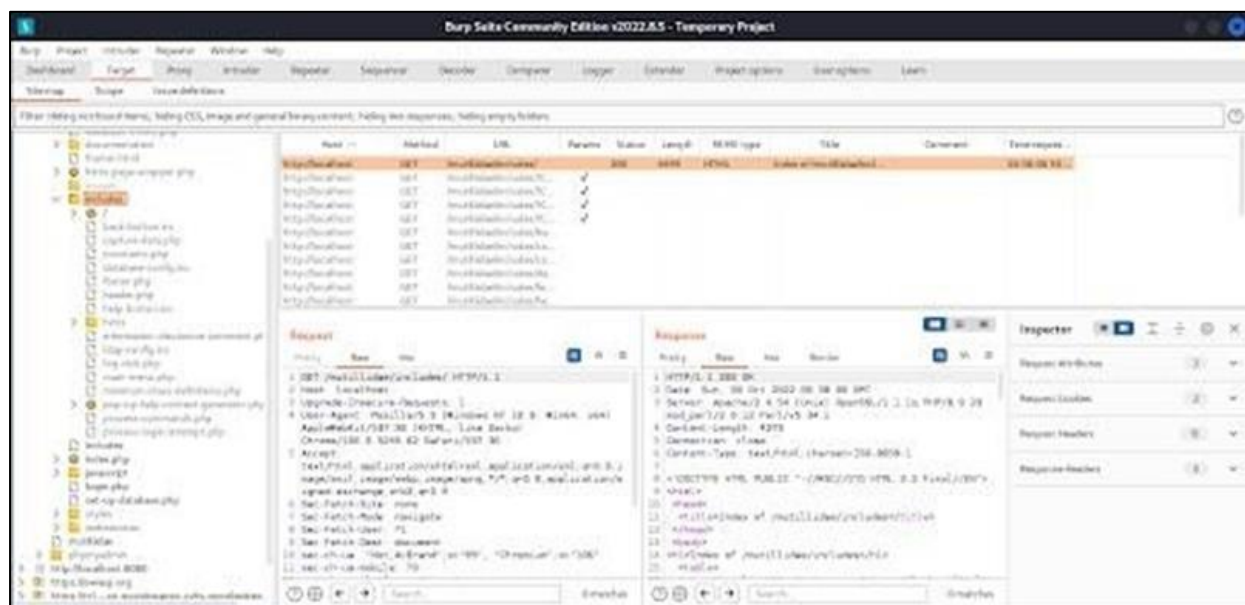
- Username:** A text input field containing 'pablo-escobar'.
- Password:** A password input field with masked characters '*****'.
- Sign in:** A blue button.
- Register new user:** A link below the 'Sign in' button.

5. Go to login screen and login to webgoat

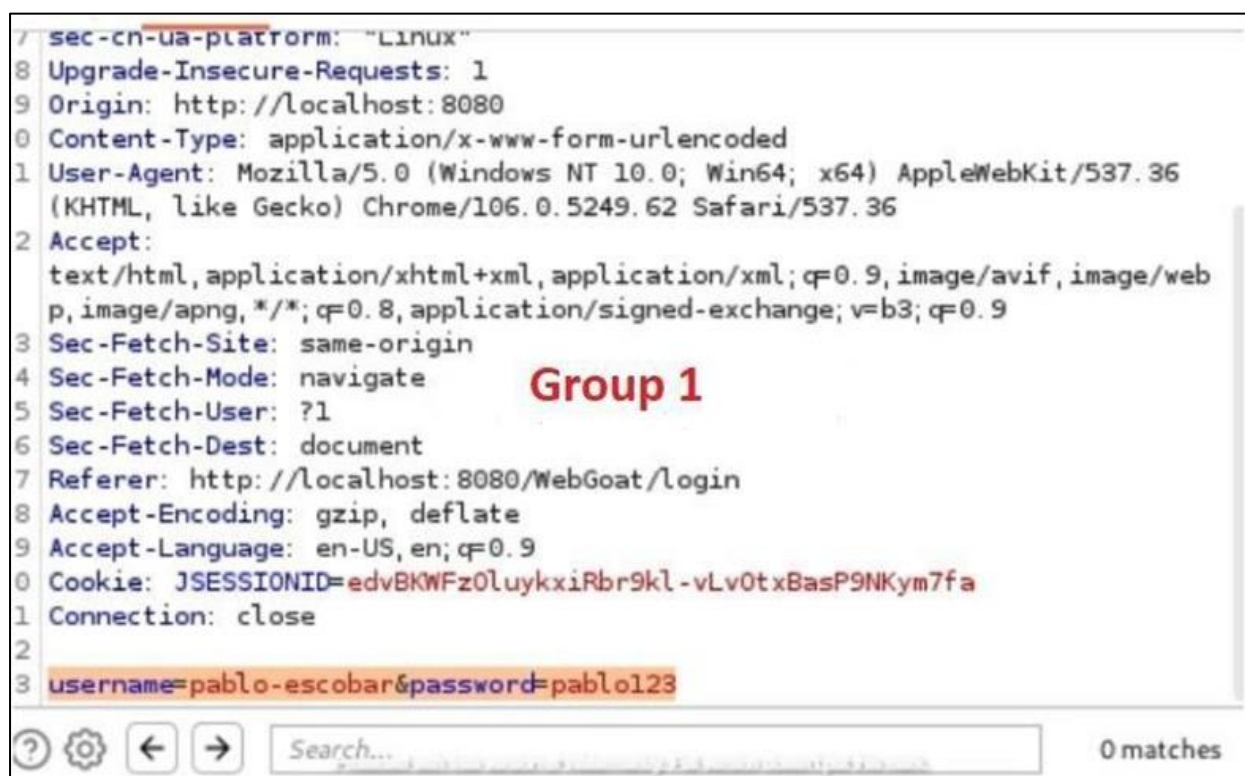


A second screenshot of the same WebGoat login page, identical to the one above. It shows the browser window with the address 'localhost:8080/WebGoat/login' and the login form containing the username 'pablo-escobar', a masked password, and the 'Sign in' button and 'Register new user' link.

6. Navigate to Proxy and then HTTP history and check for the login request



7. In this find the Session id i.e. JSESSIONID = K8xT8uiSZPi6AzLYTDtbDLm7W_EosJHN4rCJ y0BP



Defense Mechanisms Against Broken Authentication Attack:

1. Multi-Factor Authentication (MFA):

Description: Require multiple forms of authentication (e.g., password, SMS code, biometrics) to access accounts.

Implementation: Implement MFA across systems to add an extra layer of security beyond passwords.

2. Secure Password Policies:

Description: Enforce strong password policies to prevent easy-to-guess passwords and password reuse.

Implementation: Implement password complexity requirements (length, special characters, numbers) and enforce regular password changes.

3. Session Management and Expiration:

Description: Manage user sessions securely and enforce session expiration to reduce the window of opportunity for attackers.

Implementation: Implement session timeouts, token-based authentication, and re-authentication for sensitive actions.

4. Brute Force Protection:

Description: Protect against brute force attacks by limiting the number of login attempts and implementing account lockout mechanisms.

Implementation: Implement rate-limiting and temporary lockouts after a specified number of failed login attempts.

5. Proper Error Handling:

Description: Avoid exposing sensitive information through error messages that could assist attackers in understanding authentication failures.

Implementation: Customize error messages to provide minimal information to users and hide system details that could aid attackers.

6. Secure Credential Storage:

Description: Safely store user credentials using hashing algorithms (bcrypt, Argon2) combined with salting to prevent password leaks.

Implementation: Avoid storing plaintext passwords and use industry-standard encryption methods for credential storage.

7. Account Monitoring and Anomaly Detection:

Description: Continuously monitor user account activities and detect unusual behavior or login patterns.

Implementation: Use AI/ML-based anomaly detection systems to identify suspicious login attempts or activities.

8. Regular Security Assessments:

Description: Perform periodic security assessments, including vulnerability scanning and penetration testing.

Implementation: Conduct regular audits to identify vulnerabilities and weaknesses in authentication mechanisms.

Security misconfiguration attack

Security misconfiguration vulnerabilities take place when an application component is vulnerable to attack as a result of insecure configuration option or misconfiguration.

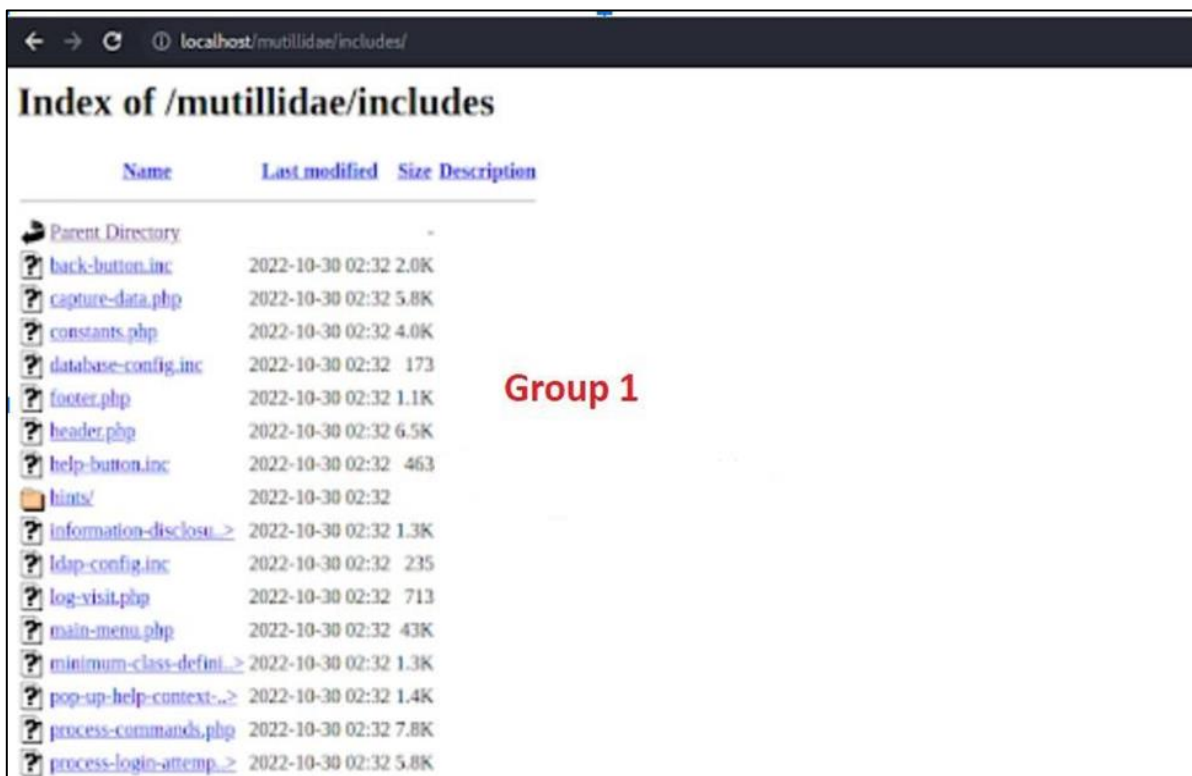
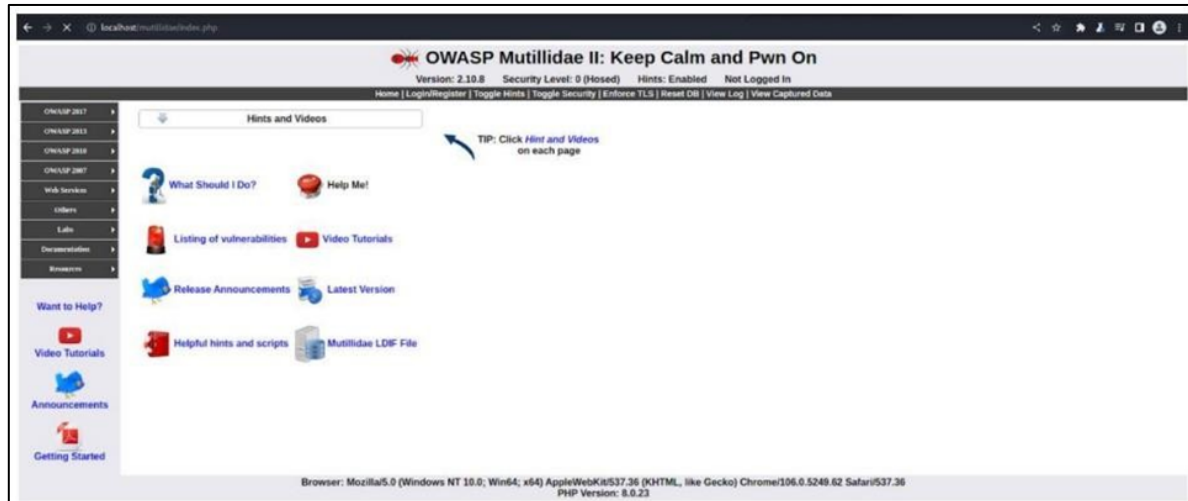
Misconfiguration vulnerabilities are configuration weaknesses that might exist in software subsystems or components. For instance, web server software might ship with default user accounts that a cybercriminal could utilize to access the system, or the software might have a known set of standard configuration files or directories, which a cybercriminal could exploit.

For instance, the following types of attacks could exploit misconfiguration vulnerabilities:

- Code injection
- Credential stuffing/brute force
- Buffer overflow • Cross-site scripting (XSS)
- Command injection
- Forceful browsing

Steps to perform security misconfiguration attack:

1. Install and run mutillidae II



2. Navigate to the site map in the target section.

The screenshot displays the Burp Suite Community Edition v2022.8.5 interface. The 'Site map' tab is active, showing a tree view of the website's structure. The 'includes' folder is expanded, revealing sub-folders like 'back-button.inc', 'captcha-data.php', 'constants.php', 'database-config.inc', 'footer.php', 'header.php', 'help-button.inc', 'info', 'information-disclosure-comment.php', 'ldap-config.inc', 'log-usb.php', 'main-menu.php', 'minimum-class-definitions.php', 'pop-up-help-content-generator.php', 'process-commands.php', and 'process-login-attempt.php'. The 'includes' folder is highlighted in orange. The 'HTTP history' tab is also visible, showing a list of requests. The first request is a GET request to 'http://localhost/includes/index.php' with a status of 200 and a length of 4495. The 'Inspector' tab is open, showing the raw HTTP request and response. The request is a GET request to 'http://localhost/includes/index.php' with a status of 200. The response is an HTML document with a title of 'Index of /includes/' and a content-type of 'text/html; charset=ISO-8859-1'.

Host	Method	URL	Params	Status	Length	MIME type	Title	Comment	Time request...
http://localhost	GET	http://localhost/includes/index.php		200	4495	HTML	Index of /includes/		02:38:08.30...

```
1 GET /includes/index.php HTTP/1.1
2 Host: localhost
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.5249.62 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/svg+xml,*/*;q=0.8,application/signed-exchange;v=b3;q=0.5
6 Sec-Fetch-Site: none
7 Sec-Fetch-Mode: navigate
8 Sec-Fetch-User: ?1
9 Sec-Fetch-Dest: document
10 sec-ch-ua: "Not A Brand";v="99", "Chrome";v="106"
11 sec-ch-ua-mobile: ?0
```

```
1 HTTP/1.1 200 OK
2 Date: Sun, 30 Oct 2022 06:38:08 GMT
3 Server: Apache/2.4.54 (Ubuntu) OpenSSL/1.1.1q PHP/8.0.23
4 Content-Length: 4273
5 Connection: close
6 Content-Type: text/html; charset=ISO-8859-1
7
8 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
9 <html>
10 <head>
11 <title>Index of /includes/
12 </head>
13 <body>
14 <div>Index of /includes/
15 </div>
```

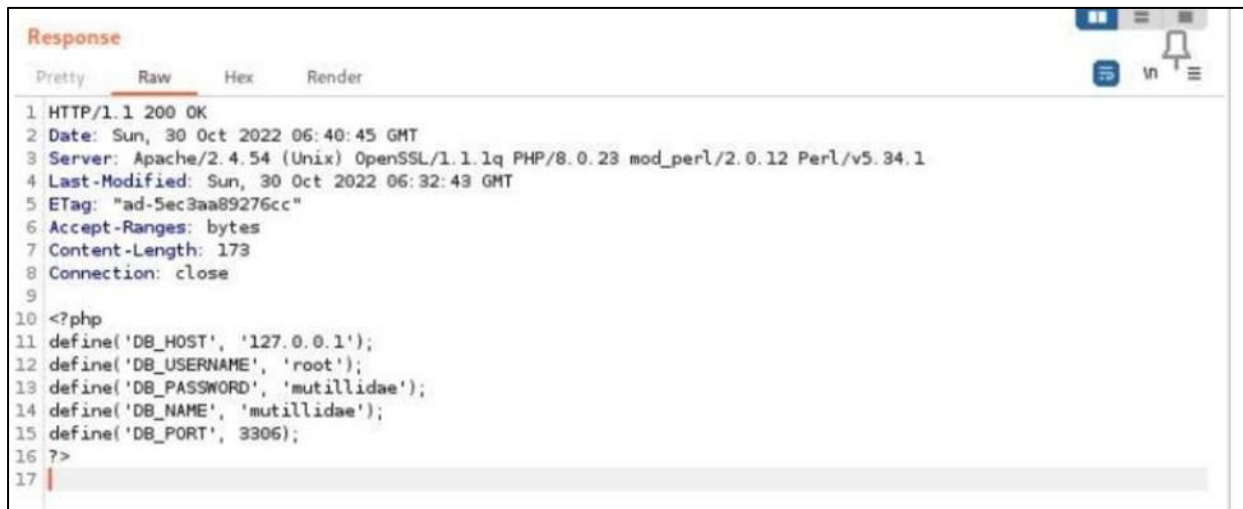
The screenshot displays the Burp Suite Community Edition v2022.8.5 interface. The 'Site map' tab is active, showing a tree view of the website's structure. The 'includes' folder is expanded, revealing sub-folders like 'back-button.inc', 'captcha-data.php', 'constants.php', 'database-config.inc', 'footer.php', 'header.php', 'help-button.inc', 'info', 'information-disclosure-comment.php', 'ldap-config.inc', 'log-usb.php', 'main-menu.php', 'minimum-class-definitions.php', 'pop-up-help-content-generator.php', 'process-commands.php', and 'process-login-attempt.php'. The 'includes' folder is highlighted in orange. The 'HTTP history' tab is also visible, showing a list of requests. The first request is a GET request to 'http://localhost/includes/index.php' with a status of 200 and a length of 4495. The 'Inspector' tab is open, showing the raw HTTP request and response. The request is a GET request to 'http://localhost/includes/index.php' with a status of 200. The response is an HTML document with a title of 'Index of /includes/' and a content-type of 'text/html; charset=ISO-8859-1'.

Host	Method	URL	Params	Status	Length	MIME type	Title	Comment	Time request...
http://localhost	GET	http://localhost/includes/index.php		200	4495	HTML	Index of /includes/		02:38:08.30...

```
1 GET /includes/index.php HTTP/1.1
2 Host: localhost
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.5249.62 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/svg+xml,*/*;q=0.8,application/signed-exchange;v=b3;q=0.5
6 Sec-Fetch-Site: none
7 Sec-Fetch-Mode: navigate
8 Sec-Fetch-User: ?1
9 Sec-Fetch-Dest: document
10 sec-ch-ua: "Not A Brand";v="99", "Chrome";v="106"
11 sec-ch-ua-mobile: ?0
```

```
1 HTTP/1.1 200 OK
2 Date: Sun, 30 Oct 2022 06:38:08 GMT
3 Server: Apache/2.4.54 (Ubuntu) OpenSSL/1.1.1q PHP/8.0.23
4 Content-Length: 4273
5 Connection: close
6 Content-Type: text/html; charset=ISO-8859-1
7
8 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
9 <html>
10 <head>
11 <title>Index of /includes/
12 </head>
13 <body>
14 <div>Index of /includes/
15 </div>
```

3. Here we can see that inside the includes folder there is one file named database-config.inc which contains database username and password without any encryption which makes it vulnerable because anyone who gets hold of this file may get important personal details of that user.



```
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Sun, 30 Oct 2022 06:40:45 GMT
3 Server: Apache/2.4.54 (Unix) OpenSSL/1.1.1q PHP/8.0.23 mod_perl/2.0.12 Perl/v5.34.1
4 Last-Modified: Sun, 30 Oct 2022 06:32:43 GMT
5 ETag: "ad-5ec3aa89276cc"
6 Accept-Ranges: bytes
7 Content-Length: 173
8 Connection: close
9
10 <?php
11 define('DB_HOST', '127.0.0.1');
12 define('DB_USERNAME', 'root');
13 define('DB_PASSWORD', 'mutillidae');
14 define('DB_NAME', 'mutillidae');
15 define('DB_PORT', 3306);
16 ?>
17
```

Defense Mechanisms Against Security Misconfigurations Attack:

1. Regular Security Audits and Reviews:

Description: Conduct regular security audits and reviews of system configurations, including operating systems, web servers, databases, and applications.

Implementation: Schedule periodic reviews and audits by security professionals to identify and rectify misconfigurations.

2. Use Security Best Practices and Standards:

Description: Follow security best practices and adhere to industry standards (such as CIS Benchmarks, OWASP guidelines, or vendor-specific hardening guides).

Implementation: Apply the latest security updates, patches, and configuration guidelines provided by software vendors.

3. Automated Configuration Management:

Description: Utilize automated configuration management tools (e.g., Ansible, Chef, Puppet) to maintain consistent and secure configurations across systems.

Implementation: Use automation to enforce standardized configurations and eliminate human errors.

4. Least Privilege Principle:

Description: Apply the principle of least privilege by granting users and systems only the necessary access and permissions.

Implementation: Regularly review access controls, permissions, and roles to ensure they are minimal and essential for operations.

5. Error Handling and Logging:

Description: Implement proper error handling and logging mechanisms to detect and respond to misconfigurations or abnormal behaviors.

Implementation: Monitor logs for unusual activities or error messages that could indicate misconfigurations or unauthorized access attempts.

6. Disable Default Accounts and Services:

Description: Disable or remove default accounts, services, and unnecessary features that may pose security risks.

Implementation: Turn off or uninstall services that are not required for the system's functionality.

7. Secure Cloud Configuration:

Description: Configure cloud services securely by following cloud provider security guidelines and recommendations.

Implementation: Utilize the built-in security features and tools provided by cloud service providers and configure them properly.

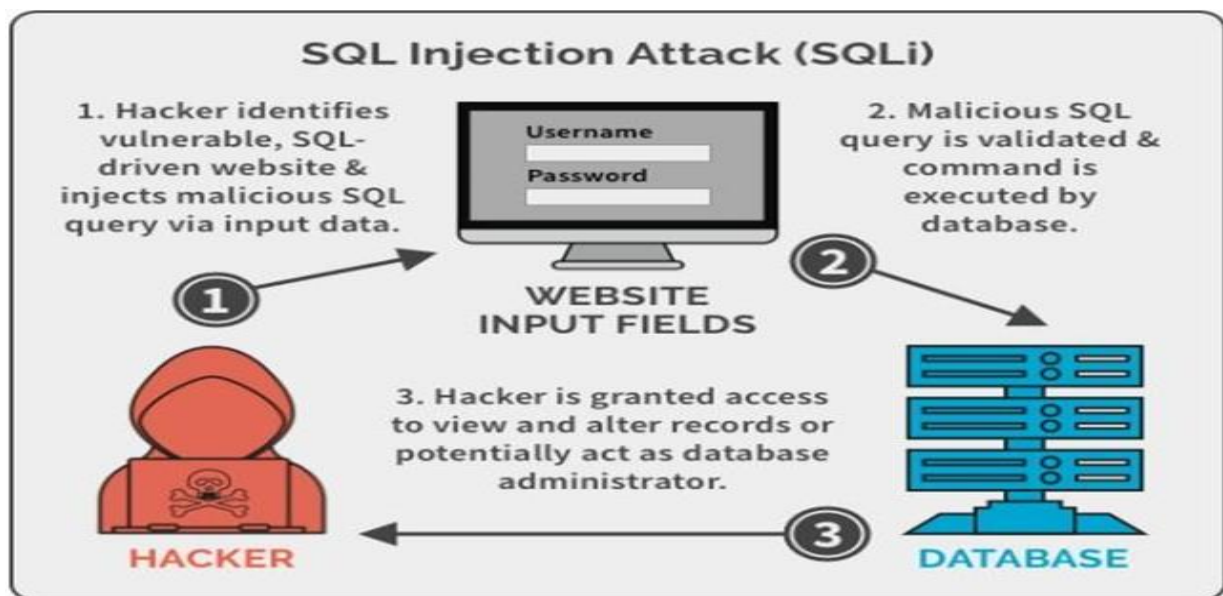
8. Regular Backups and Recovery Plans:

Description: Implement regular backups of critical data and create recovery plans to restore systems to a secure state.

Implementation: Test backup restoration processes periodically to ensure data integrity and system recovery readiness.

Injection attack

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content

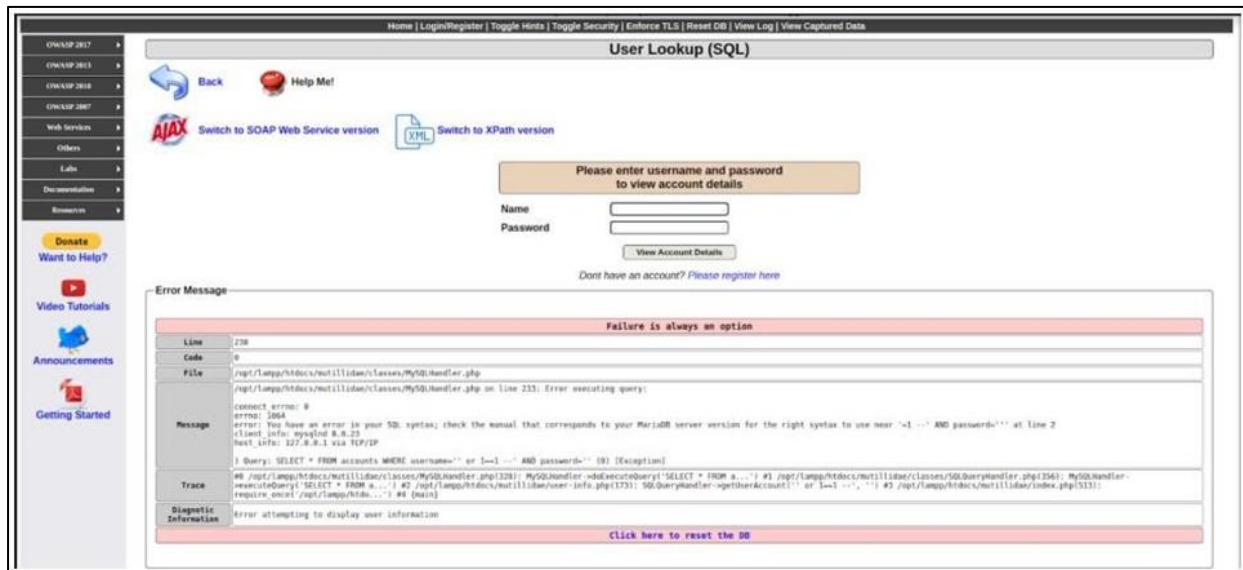


Steps to perform injection attack:

1. Run the vulnerable website



2. Error message giving away too much information about the database



3. Write admin'#

Please enter username and password to view account details

Name

admin'#

Password

View Account Details

Dont have an account? [Please register here](#)

Results for "admin'#". 1 records found.

Username=admin
Password=adminpass
Signature=g0t r00t?

4. Here we can see that we were able to smartly input a string in the username field which can break the sql logic. We are successfully able to fool the database into giving us admin credentials.

5. Let's now try creating a new user account and try to get its credentials.

Please choose your username, password and signature

Username

Password

[Password Generator](#)

Confirm Password

Signature

Create Account


**Posted Token:
(Validation not performed)**


Expected Token For This Request:

Token Passed By User For This Request:

New Token For Next Request:

Token Stored in Session:

 Switch to SOAP Web Service version

 Switch to XPath version

**Please enter username and password
to view account details**

Name

Password

View Account Details

Dont have an account? [Please register here](#)

Results for "Pablo'#" .1 records found.

Username=Pablo

Password=pablo_vjti

Signature=Hello, I am Pablo Escobar from VJTI

Defense Mechanisms Against Injection Attack:

1. Parameterized Queries (Prepared Statements):

Description: Use parameterized queries with placeholders instead of dynamically constructing SQL queries.

Implementation: Utilize frameworks and libraries that support parameter binding to separate data from the query itself.

2. Input Validation and Sanitization:

Description: Validate and sanitize user inputs to ensure they adhere to expected formats and contain no malicious content.

Implementation: Use input validation routines and allowlisting (accept only known good inputs) to discard or encode malicious characters.

3. Escaping and Encoding:

Description: Escape user inputs before using them in queries or commands to neutralize special characters.

Implementation: Employ escaping functions or encoding techniques appropriate for the context (e.g., HTML encoding, SQL escaping).

4. Least Privilege Principle:

Description: Restrict access rights and permissions to limit the impact of a successful injection attack.

Implementation: Ensure that database users and application accounts have minimal privileges required for their specific functions.

5. Use of Stored Procedures:

Description: Utilize stored procedures to encapsulate logic within the database and reduce the need for dynamic SQL queries.

Implementation: Implement business logic in stored procedures, allowing inputs to be treated as parameters rather than executable code.

6. Web Application Firewalls (WAFs):

Description: Deploy WAFs to monitor and filter HTTP/HTTPS requests for malicious payloads.

Implementation: Configure WAFs to detect and block suspicious inputs that could be indicative of injection attacks.

7. Regular Security Patching:

Description: Keep database systems and application frameworks up-to-date with the latest security patches.

Implementation: Monitor for security advisories and promptly apply patches to fix known vulnerabilities that could be exploited.

8. Static and Dynamic Code Analysis:

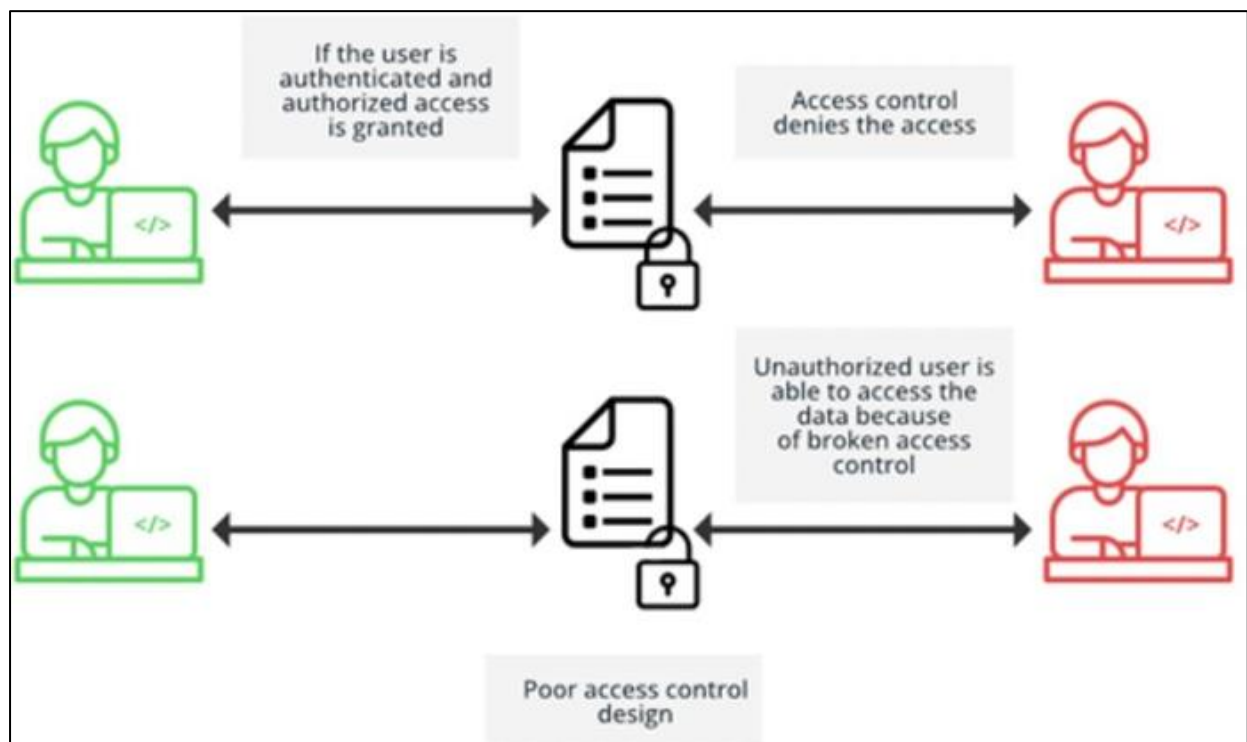
Description: Perform static analysis to identify vulnerabilities in source code and dynamic analysis to simulate attack scenarios.

Implementation: Utilize automated tools for code scanning and penetration testing to detect and remediate injection vulnerabilities.

Broken access control attack

Access control, or authorization, is how a web application grants users access to some resources, but not others. These resources mostly fall into two categories: sensitive data, which should only be accessed by select users,

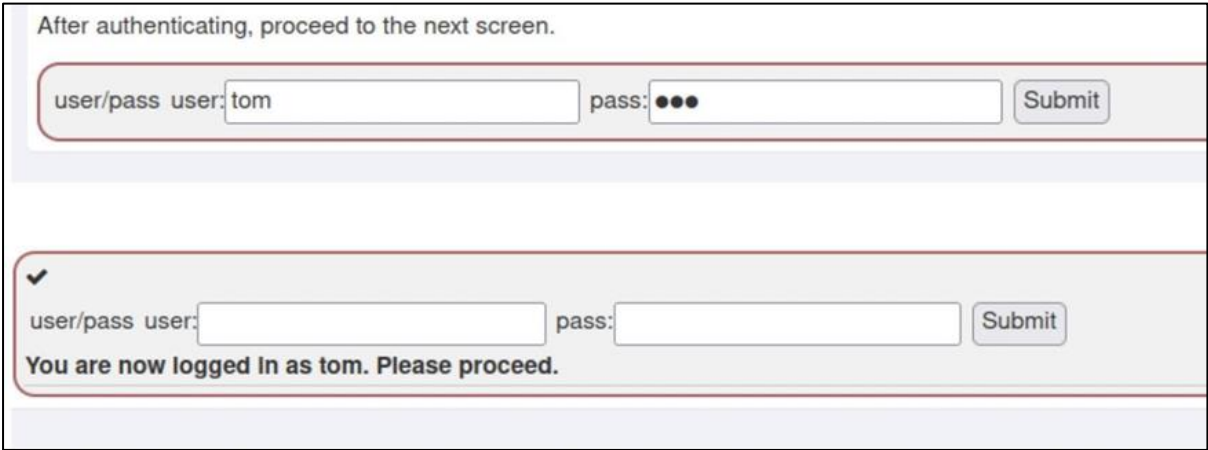
and functions that can modify data on the webserver, or even modify the server's functionality. Authorization checks are performed after authentication: when a user visits a webpage, they must first authenticate themselves.



Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits. Common access control vulnerabilities include: Violation of the principle of least privilege or deny by default, where access should only be granted for particular capabilities, roles, or users, but is available to anyone.

Steps to perform broken access control

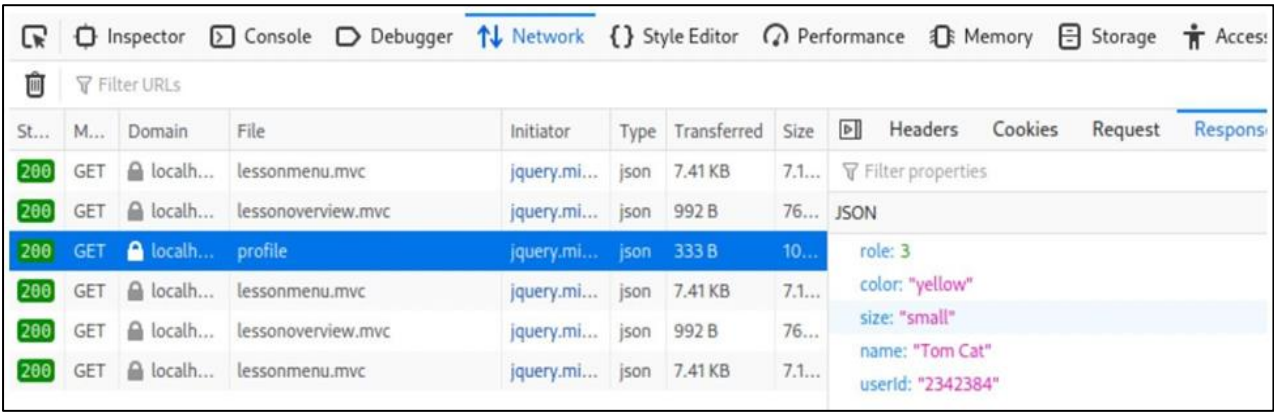
- 1. Install webgoat and run and Login into webgoat



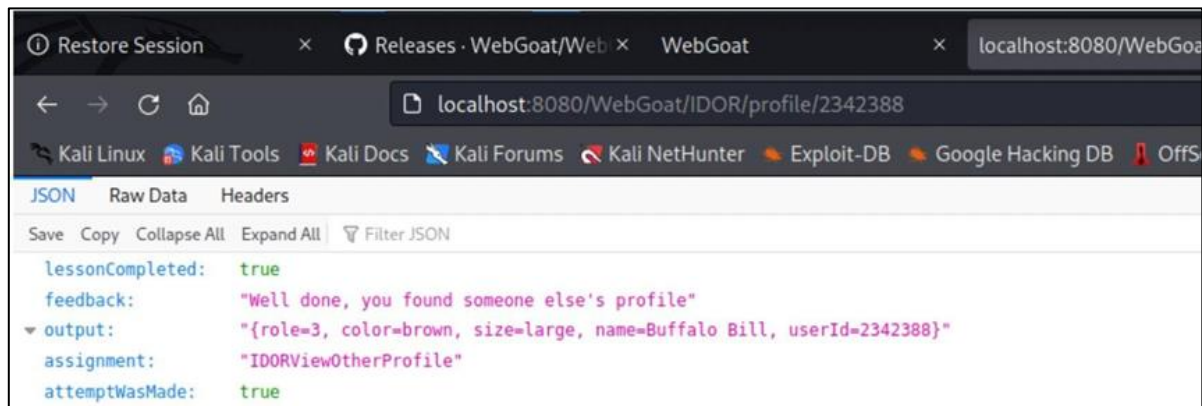
- 2. We can view our profile by clicking on that box



- 3. We can check the network history of that site and find the profile request that was made



4. From here we can see our userid and also the api call that was made to get the profile details and from which we can get profile details of other users too by brute force method i.e. trying any userid.



5. So after applying brute force method we were able to get the details of a user with userid 2342388 and name Buffalo Bill.

Defense Mechanisms Against Broken Access Control Attack:

1. Role-Based Access Control (RBAC):

Description: Implement a role-based access control model to assign permissions based on roles or groups.

Implementation: Define roles with specific privileges and assign users to these roles to limit access based on their responsibilities.

2. Principle of Least Privilege (PoLP):

Description: Apply the principle of least privilege, granting users only the minimum permissions required for their tasks.

Implementation: Regularly review and refine access permissions to ensure users have the minimum necessary access.

3. Access Control Lists (ACLs):

Description: Use access control lists to explicitly specify who can access particular resources or perform specific actions.

Implementation: Apply ACLs at the object level to control access to files, directories, or database records.

4. Attribute-Based Access Control (ABAC):

Description: Leverage attributes such as user attributes, environmental conditions, or resource attributes for access decisions.

Implementation: Define policies that consider various attributes to make access control decisions dynamically.

5. Regular Access Reviews:

Description: Conduct periodic access reviews and audits to ensure that access permissions are appropriate and up-to-date.

Implementation: Perform regular checks to revoke or modify unnecessary or outdated access rights.

6. Secure Direct Object References (IDOR Prevention):

Description: Prevent direct object references by using indirect references or tokens to access resources.

Implementation: Utilize unique identifiers or tokens that are meaningless outside the application context to access objects.

7. Session Management and Authentication Tokens:

Description: Secure session management and authentication tokens to prevent session hijacking and unauthorized access.

Implementation: Implement secure session handling, enforce session expiration, and use secure token generation mechanisms.

8. Access Control Testing:

Description: Regularly test access controls to identify and address vulnerabilities and misconfigurations.

Implementation: Conduct security assessments, penetration testing, and use tools to detect and fix access control weaknesses.

Conclusion

Thus, in this experiment we got to know the top 10 OWASP attacks, its preventive measures and also performed 4 out of that 10 attacks using kali linux, webgoat, mutillidae and burpsuite.