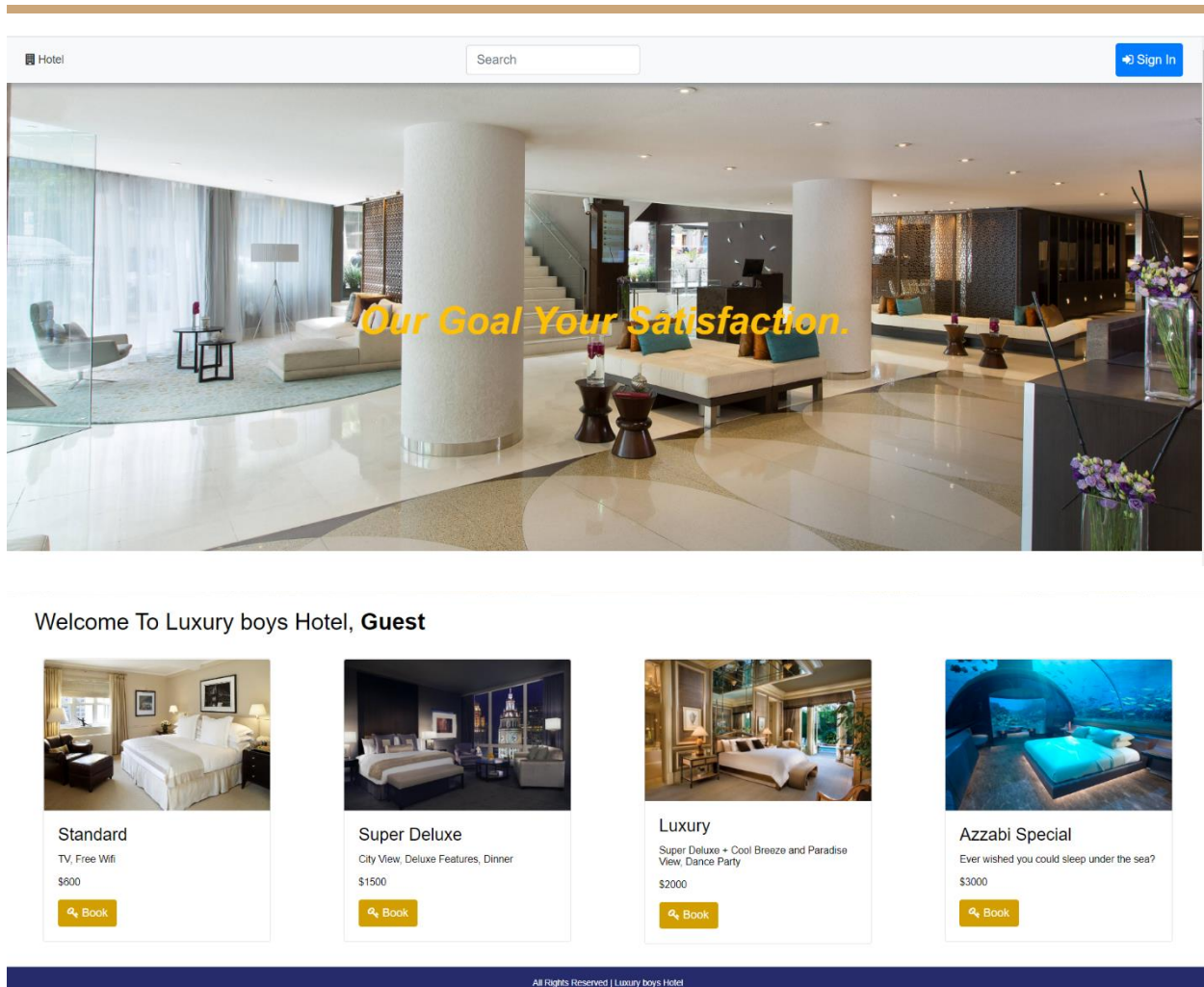


Hotel BOOKING WEBSITE

BY: Ali Abdelhamid, Ramy Naiem, Adham ALAzzabi, Waleed Ibrahim



Abstract

This is a web-based Booking website that is published online using a premium DNS, we are going to assess the code as well as use multiple vulnerability scanners such as W3af, Nikto, and owasp zap in order to detect all weaknesses and threats present then implement a mitigation plan to prevent malicious users from penetrating the website and protect our obscure information, Furthermore, we will mention how we prevented attacks such as DDOS attacks anti-DDoS fortified system, SQL injection, etc. We will then form a report that includes all the steps we took to accomplish the creation of a highly secure website as well as the creation of the code and every member's contribution to the project.

Table of Contents

Abstract	1
Time plan	2
Web application description:.....	3
Milestone 2	4
SQL injection	5
cross-site scripting (XSS)	6
Blind OS.....	7
Access control	8
Methodology for pen-testing.....	9
Mitigation techniques	10
Security features implemented on the register page:.....	11
Security features implemented on the login page:	12
Security features implemented on the main index page:	13
Security features implemented on the logout page:.....	14
Vulnerability Assessment:.....	15

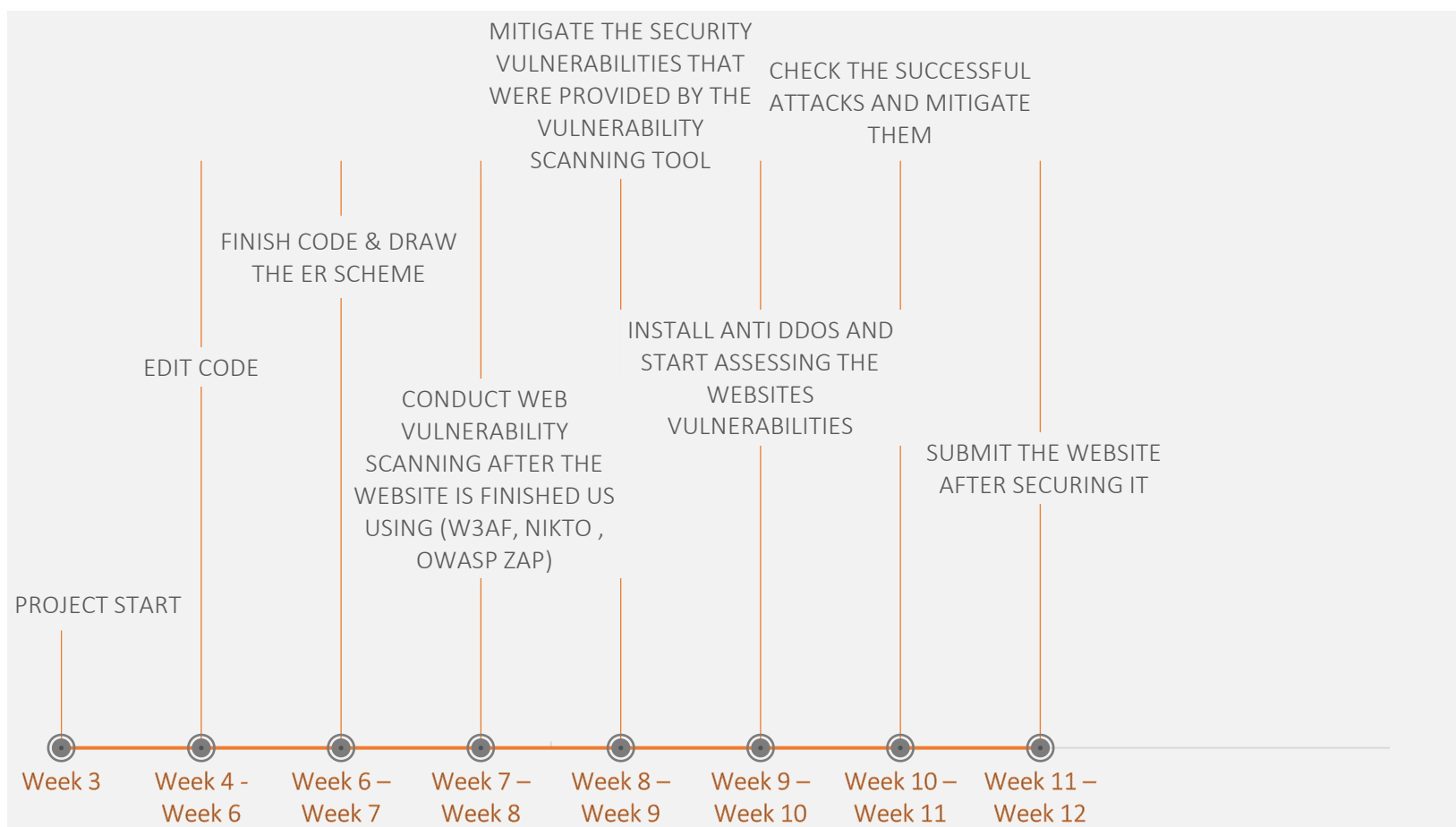
Time plan

Weeks	Weeks Activities
Week 4 – Week 6	<ul style="list-style-type: none">• Finish the Database• Finish the Code (HTML & CSS)• Edit PHP & JS part• Start Outlining the report
Week 6 – Week 7	<ul style="list-style-type: none">• Finish the PHP & JS• Draw the ER diagram• Identify the relationship and the primary key• Draw the ER scheme
Week 7 – Week 8	<ul style="list-style-type: none">• Conduct web vulnerability scanning after the website is finished using these tools (W3af, Nikto , owasp zap ...etc.)• Editing the report and adding the new milestones
Week 8 – Week 9	<ul style="list-style-type: none">• Mitigate the security vulnerabilities that were provided by the vulnerability scanning tool• Implementing security features• Secure the code• Encrypt & hash the database
Week 9 – Week 10	<ul style="list-style-type: none">• Installing a fortified anti DDoS system and test it using multiple attack methods from multiple directions.
Week 10 – Week 11	<ul style="list-style-type: none">• check the successful attacks and mitigate them• Conduct vulnerability assessment and check website efficiency• simulate multiple attacks such as (SQL injection, DDoS attack... etc.)

Week 11 – Week 12

- final check the successful attacks and mitigate them
- produce the final report
- submit the website after securing it
- submit CW final report

Web Security Project Timeline graph

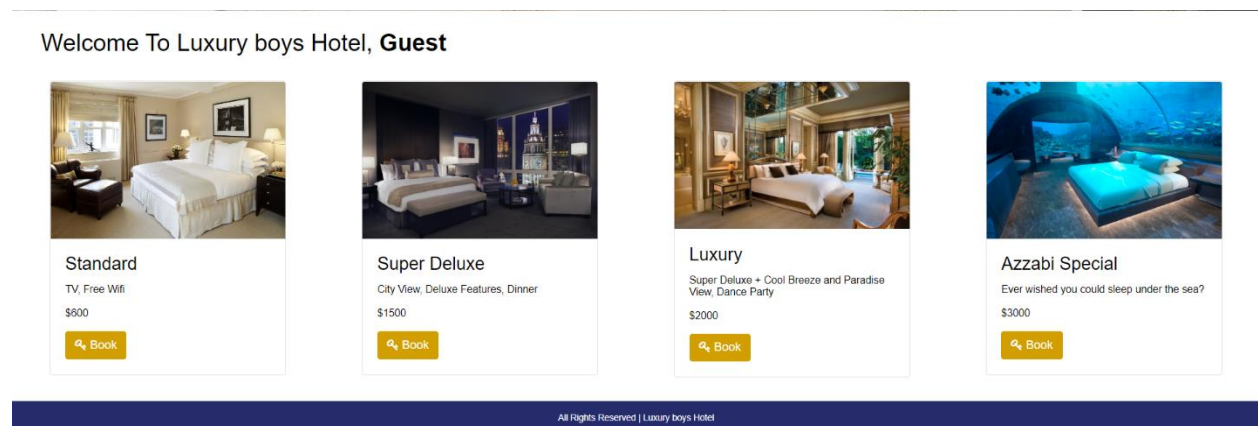


Web application description:

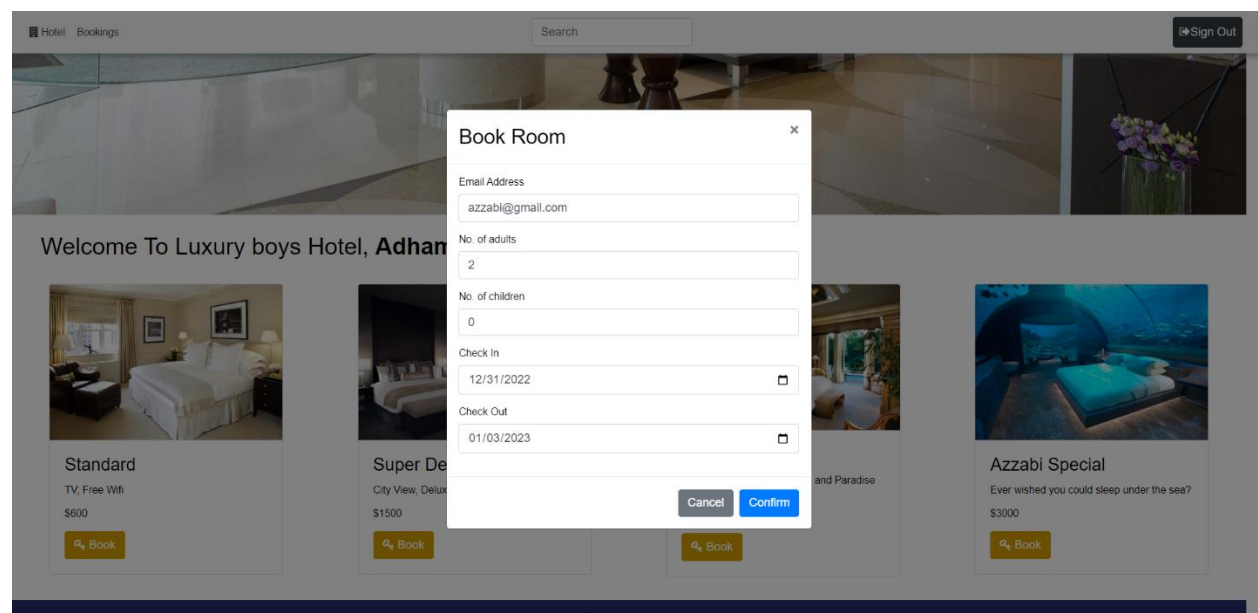
Features of the web-based application


The user will provide the interface with an email then establish a password. It is then required to log in using the newly generated credentials.


The platform's home page, often known as the homepage or home screen, is where users can choose to book from the various available rooms.



Most of what the home page shows are rooms that are available to book. The user can see the available rooms with its price information shown. Prices might vary on the size of the room or location. After clicking on the room of choice, the user can then pick their desired check-in day of the booking and the check-out date. The user will also have the option to see all the bookings that he or she has made and can delete any if needed. There is also the option of looking up a room using its name.





[Hotel](#)
[Bookings](#)


[Sign Out](#)

Luxury boys Hotel - Bookings

Booking No.	Room No.	Customer Email	No. of guests	Check-in Date	Check-Out Date	Action
36	1	aazzab622@gmail.com	2	2022-11-25	2022-11-24	Edit Delete
38	5	aazzab622@gmail.com	2	2022-11-20	2022-11-23	Edit Delete

All Rights Reserved | Luxury boys Hotel

<div> <div>  Hotel </div> <div> Rooms </div> <div> Bookings </div> </div> <div> <input type="text" value="Search"/> </div> <div> Sign Out </div>				
<div> <div>Luxury boys Hotel - Rooms</div> <div>Add New</div> </div>				
Prod No.	Room Type	Price	Description	Action
1	Standard	600	TV, Free Wifi	<div>EditDelete</div>
4	Super Deluxe	1500	City View, Deluxe Features, Dinner	<div>EditDelete</div>
5	Luxury	2000	Super Deluxe + Cool Breeze and Paradise View, Dance Party	<div>EditDelete</div>
6	Azzabi Special	3000	Ever wished you could sleep under the sea?	<div>EditDelete</div>
<div>All Rights Reserved Luxury boys Hotel</div>				

Hotel

Rooms

Bookings

Search

Sign Out

Luxury boys Hotel - Bookings

Booking No.	Room No.	Customer Email	No. of guests	Check-In Date	Check-Out Date	Action
36	1	azzab622@gmail.com	2	2022-11-25	2022-11-24	<div>EditDelete</div>
37	6	trial1@gmail.com	2	2022-11-13	2022-11-27	<div>EditDelete</div>
38	5	azzab622@gmail.com	2	2022-11-20	2022-11-23	<div>EditDelete</div>
39	6	new@gmail.com	2	2022-12-31	2023-01-02	<div>EditDelete</div>
40	4	A@gmail.com	2	2022-12-14	2023-01-04	<div>EditDelete</div>
41	4	f@gmail.com	2	2022-12-31	2023-01-03	<div>EditDelete</div>

All Rights Reserved | Luxury boys Hotel

Milestone 2

There are a lot of security concerns when it comes to web applications, therefore in this coursework, we will be discussing some of the different security concerns, also implementation techniques will be considered.

The vulnerabilities listed will be divided between the team. As this will ensure that each aspect of the web application is secure.

SQL injection

- Union & Blind SQL attack - SQL UNION Injection attacks

cross-site scripting (XSS)

- DOM XSS in document.write sink using source location.search inside a select element
- Exploiting cross-site scripting to steal cookies or to capture passwords
- Reflected XSS into HTML context with most tags and attributes blocked
- Stored XSS into onclick event with angle brackets and double quotes
- HTML encoded and single quotes and backslash-escaped

Blind OS

- Blind OS command injection with time delays/output redirection / out-of-band data exfiltration

Access control

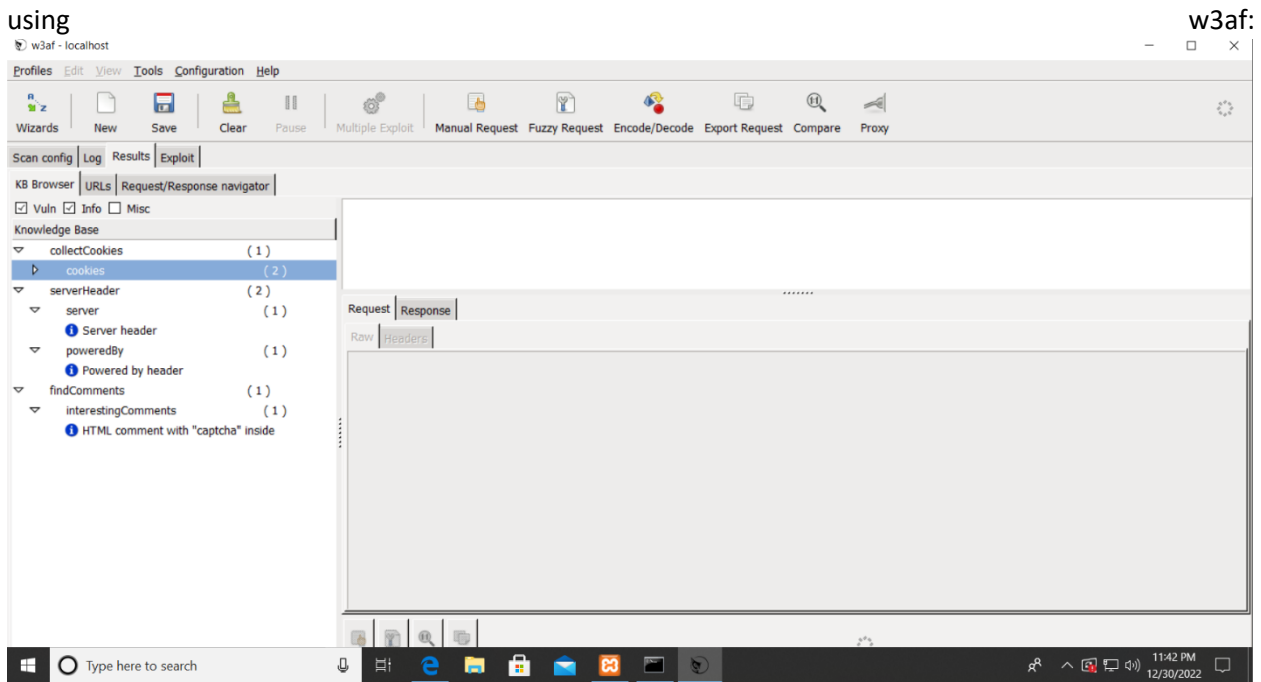
- User role can be modified in the user profile
- URL-based access control can be circumvented
- Method-based access control can be circumvented
- User ID controlled by request parameter, with unpredictable user IDs
- User ID controlled by request parameter with data leakage in redirect
- User ID controlled by request parameter with password disclosure
- Username enumeration via (different responses - subtly different responses - response timing)
- Broken access control
- Insecure design

Methodology for pen-testing

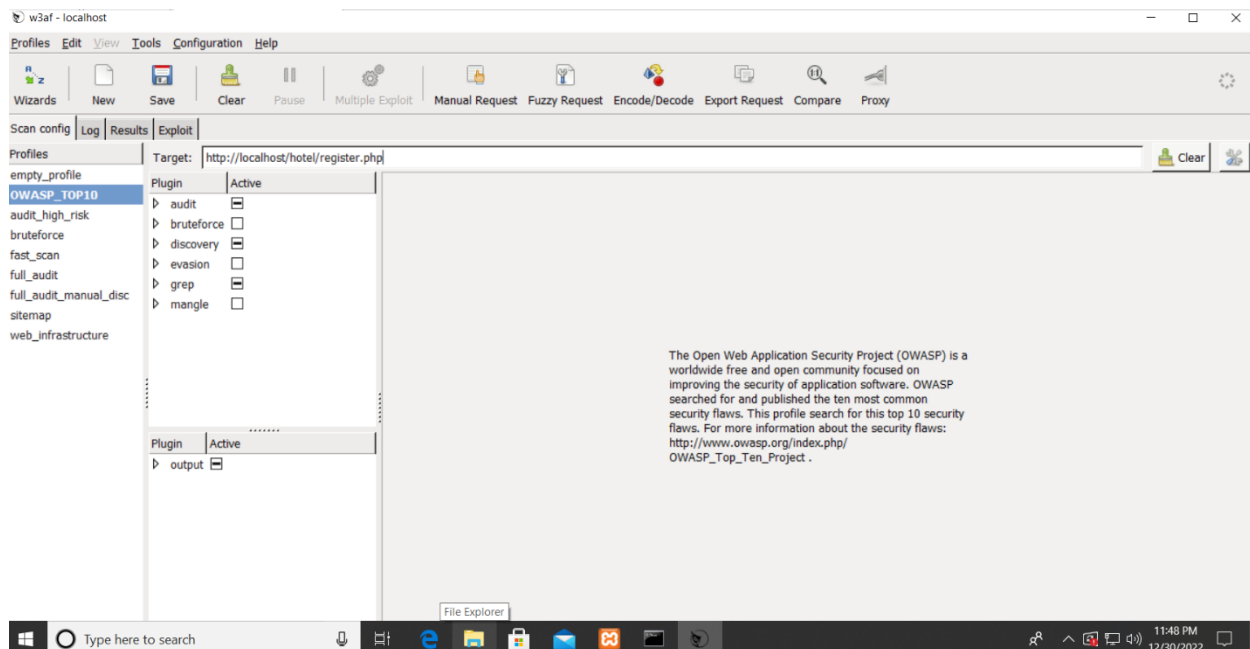
To fully understand and mitigate the attacks that were mentioned above, there are some steps that will be taken by each team member:

Firstly, we will be reviewing all port swigger labs that are related to these vulnerabilities and deeply understand how they are conducted and how they are mitigated. Also, all the group members will start testing and attacking the website using the labs as well as using pen testing applications such as hydra to brute force and test the website even more.

Group members will also utilize web security assessment tools such as w3af, Nikto, Owasp zap on a regular basis to produce a comprehensive summary containing possible risks. Examples of the scans completed using



This scan was done on the login.php page and there were no vulnerabilities found after successfully mitigating all the attacks



This scan was done on the register.php. as you can see there was no vulnerabilities at all found, this is because the tests were done on the final version of the website. The version we have secured after all the testing we did.

Mitigation techniques

Security features implemented on the register page:

- 1- Escaping special characters: The script uses the `mysqli_real_escape_string()` function to escape any special characters in user input. This helps prevent SQL injection attacks by ensuring that any special characters in user input are treated as literal characters rather than being interpreted as part of an SQL query. We tried multiple attacks on SQL injection using port swigger and such but due to the vigorous implementations we did none of them were successful, for example, we tried `"password' OR 1=1 --"` which was escaped by the `mysqli_real_escape_string()`.
- 2- Validating email format: The script uses the `filter_var()` function to validate the email format. This helps prevent users from entering invalid or malicious emails. This was done to ensure that an actual email is being entered, protecting against malicious emails by validating the email format, you can help prevent users from entering malicious emails that may be used to attack your system. For example, users may try to enter an email address with a malicious payload, such as one that contains JavaScript code that could be used to execute an XSS attack. By validating the email format, you can help prevent these types of attacks. Ensuring

accurate data collection by validating the email format, can help ensure that the user data you collect is accurate and reliable. This can be important for a variety of purposes, such as sending emails, analyzing user behavior, or reporting on user demographics. On this feature, we tried the following string "attacker@example.com<script>alert('XSS attack')</script>" to try to bypass the email form we had set but due to the input sanitization, we utilized it did not work.

- 3- Checking if the email is already registered: The script checks if the email is already registered in the database by preparing and executing a SELECT statement. This helps prevent users from creating multiple accounts with the same email. If this was not done properly it would present a multitude of issues, such as confusion, for the user as it would be difficult to know which account the user wants to log in to use the same email. As well as security issues, if multiple users have the same email, it can be difficult to determine which user is the owner of the email. This can lead to security issues, as someone may be able to gain access to an account, to which they shouldn't have access. In this following point, we tried entering "' OR 1=1; -- attacker@example.com" in the email field in order to enter an email that has been used to register using before, but it did not accept the invalid email syntax.
- 4- Validating password: The script checks if the password is empty, less than 7 characters, or doesn't contain at least one letter, one number, and one symbol. This helps ensure that the password is strong and secure. We also made a very strong password policy put in place to make sure that a strong and secure password is guaranteed across all user accounts. Which will provide protection against brute force attacks, A strong password policy can help prevent brute force attacks, where an attacker tries to guess a user's password by trying many different combinations. By requiring long, complex passwords, you can make it more difficult for an attacker to successfully guess a password through a brute force attack. Protection against brute force attacks: A strong password policy can help prevent brute force attacks, where an attacker tries to guess a user's password by trying many different combinations. By requiring long, complex passwords, you can make it more difficult for an attacker to successfully guess a password through a brute force attack. We tried to brute force the password using hydra but due to the strong password policy put in place the attack was futile and would've taken us a very long period to complete. We did not try any attacks on this due to the scarce variety of attacks you can do to exploit a password validation field.
- 5- Matching password and confirm password: The script checks if the password and confirm password fields match. This helps ensure that the user entered the correct password and helps prevent mistakes in password entry. This is only put in place for the users convenience and prevent any type of confusion, for example, the user may have a typo while typing in their password which would lead to them not being able to log into their account after registering.
- 6- Verifying CAPTCHA: we integrated the captcha into the page itself in the anti DDOS we implemented, this was in order to increase user convenience as well as cut the hassle of the user having to confirm them not being a bot every time, they try to log in. Captcha was also

implemented to reduce spam and abuse by making it more difficult for automated systems to perform certain actions. For example, if you require a CAPTCHA to be solved before allowing users to post comments on a blog, it may reduce the amount of spam comments that are posted.

- 7- Anti DDos: Anti-DDoS measures are designed to protect against DDoS attacks, which are a type of cyber-attack that involve flooding a server or network with traffic from multiple sources in order to overwhelm the system and make it unavailable to users. Anti-DDoS measures work by identifying and blocking traffic that is part of a DDoS attack, while allowing legitimate traffic to pass through, helping to ensure that a website or application remains available to users even during a DDoS attack.

Security features implemented on the login page:

- 1- Input validation: The script checks if the email and password fields are empty and display an error message if they are. This helps prevent users from attempting to log in with incomplete or blank credentials. ensuring that required fields are completed by checking if the email and password fields are empty, you can ensure that the user has completed all the required fields. This can help prevent errors or problems that may arise from using incomplete or blank credentials, such as being unable to log in or reset a password. As well as improving the user experience, you can help improve the user experience by ensuring that users are not allowed to proceed with incomplete or invalid credentials. This can prevent frustration and confusion for the user and help them complete the account creation process smoothly. In the example of an attack, we tried changing from the customer role on the login page to the administrator role, but we have hidden the roles therefore no one can change it. We have also implemented that there can only be one single user with admin privileges.
- 2- CAPTCHA verification: The script verifies the CAPTCHA response using the Google reCAPTCHA API to help prevent automated bot attacks. With the aim the aim of protecting against data scraping: Automated bots can be used to scrape data from a website or application. By requiring a CAPTCHA to be solved before allowing certain actions, you can help prevent bots from scraping data from your site. In addition to, protection against distributed denial of service (DDoS) attacks: Automated bots can be used to launch distributed denial of service (DDoS) attacks, which can cause a website or application to become unavailable to users. By requiring a CAPTCHA to be solved before allowing certain actions, you can help prevent bots from launching DDoS attacks.
- 3- Prepared statements: The script uses prepared statements to execute the SQL query for selecting a user from the database. Prepared statements help prevent SQL injection attacks by separating the SQL code from the user-provided input. Offering improved security due to prepared statements having automation in escaping special characters in user-provided

input, which helps prevent attackers from using these characters to manipulate the SQL code. Prepared statements also allow you to separate the SQL code from the user-provided input by using placeholders for the input values. This means that the user-provided input is not directly included in the SQL code, which helps prevent attackers from injecting malicious code into the SQL code.

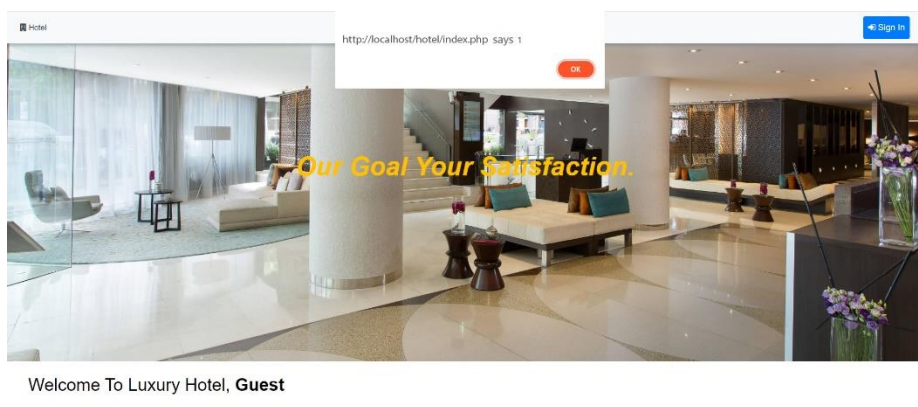
- 4- Password hashing: The password stored in the database for each user is hashed, which means that it is transformed into a random, fixed-size string that cannot be easily reversed to obtain the original password. This helps protect user passwords in case the database is compromised. Hashing passwords proved to add many security features: By hashing user passwords, you can help protect them from being accessed or stolen if the database is compromised. Even if an attacker gains access to the hashed passwords, they would not be able to easily obtain the original passwords because the hashing process is irreversible. As well as compliance with regulations as in some cases, hashing user passwords may be required by law or industry regulations. For example, certain regulations may require that personal data, including passwords, be stored in a secure and encrypted format.
- 5- Anti DDoS: (Distributed Denial of Service) measures are important because they help protect against DDoS attacks, which can cause a website or application to become unavailable to users. DDoS attacks involve flooding a server or network with traffic from multiple sources, with the goal of overwhelming the system and preventing it from being able to respond to legitimate requests. Anti-DDoS measures can help protect against these types of attacks by identifying and blocking traffic that is part of a DDoS attack, while allowing legitimate traffic to pass through. This can help ensure that a website or application remains available to users even in the face of a DDoS attack.

[Security features implemented on the main index page:](#)

- 1- Including an anti-DDoS library: A DDoS (Distributed Denial of Service) attack is a type of attack that involves overwhelming a server or network with traffic from multiple sources in order to disrupt service. An anti-DDoS library can help to protect against this type of attack by identifying and blocking malicious traffic. An anti-DDoS library can help to mitigate the effects of a DDoS attack by identifying and blocking malicious traffic before it reaches your server or network. This can help to ensure that your service remains available to legitimate users, even in the face of a DDoS attack. To confirm the function of the DDoS system put in place we launched a volumetric attack where we flooded the target with a large volume of traffic such as sending many fake requests to a web server.
- 2- Sanitizing and validating user input: This involves checking user input to ensure that it meets certain criteria and is not malicious. Sanitizing input involves removing any potentially harmful characters or code, while validating input involves checking that it meets certain requirements (e.g., a room number must be a number within a certain range). This can help to prevent injection attacks, in which an attacker injects malicious code into the application through user input. For

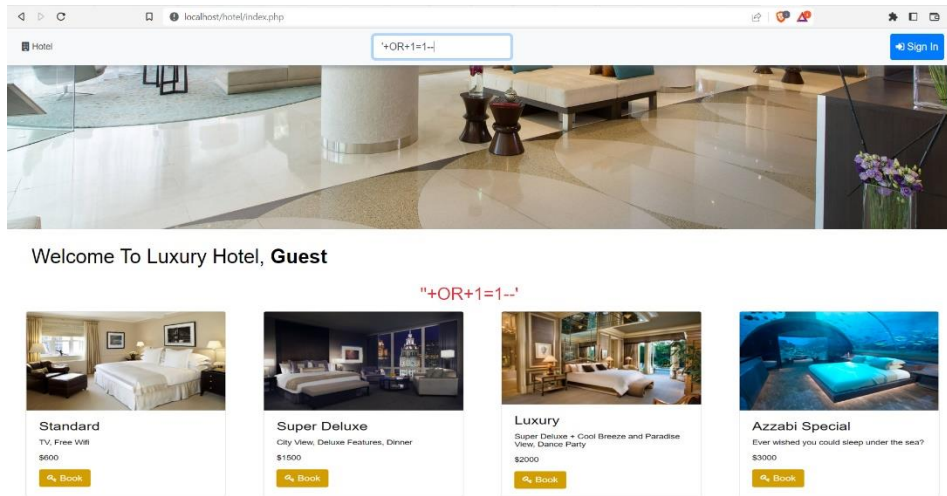
example, if the application does not properly sanitize user input, an attacker could potentially inject SQL code into a form field in order to execute arbitrary SQL queries on the database. This could potentially allow the attacker to access or modify sensitive data stored in the database.

- 3- Checking the room number against a whitelist: By checking user-provided input against a predetermined list of allowed values (a "whitelist"), you can ensure that only valid room numbers are accepted. This can help to prevent attacks in which an attacker submits invalid or malicious input to gain unauthorized access or disrupt the application. It is important to check user-provided input against a whitelist of allowed values to help prevent attacks in which an attacker submits invalid or malicious input to gain unauthorized access or disrupt the application. This is because a whitelist can help to ensure that only valid input is accepted by the application, which can help to reduce the risk of security vulnerabilities or other types of attacks. An attack we tried was "room_number = '101 OR 1=1'" using this into the room number which would allow the attacker to view all the bookings on this room.
- 4- Using prepared statements with bound parameters: Prepared statements are a way to execute SQL queries in a way that is safe against SQL injection attacks. When you use prepared statements, you write a SQL query with placeholders for the user-provided input, and then "bind" the actual input to the placeholders when the query is executed. This helps to prevent an attacker from injecting malicious code into the query through the input. Prepared statements are a way to execute SQL queries in a way that is safe against SQL injection attacks. When you use prepared statements, you write a SQL query with placeholders for the user-provided input, and then "bind" the actual input to the placeholders when the query is executed. This helps to prevent an attacker from injecting malicious code into the query through the input because the input is treated as a separate value rather than being directly included in the query. We tried an attack before we implemented the prepared statement's parameter, and this was the result:
After successfully implementing the prepared statements, the attack failed instantly.



- 5- Using htmlspecialchars and strip_tags: These functions can be used to sanitize output to prevent XSS (Cross-Site Scripting) attacks. htmlspecialchars convert certain characters to their HTML entity equivalents (e.g., < becomes <), which helps to prevent an attacker from injecting HTML or

JavaScript code into the output. `strip_tags` remove HTML and PHP tags from the output, which can also help to prevent XSS attacks. To prevent XSS attacks, it is important to sanitize the output that is generated by the application. Functions like `htmlspecialchars` and `strip_tags` can be used to sanitize output in order to prevent XSS attacks. `htmlspecialchars` convert certain characters to their HTML entity equivalents (e.g., `<` becomes `<`), which helps to prevent an attacker from injecting HTML or JavaScript code into the output. For example, if an attacker tries to inject the following code into the output: `<script>alert('XSS attack')</script>` the output would be `alert('XSS attack')` disabling any threat of XSS attacks. An attack we tried was:



After implementing the `htmlspecialchars` and `strip_tags` the exploit stopped working and the security of the website was increased.

- 6- Checking user login status: It is important to check the login status of users in order to prevent unauthorized access to certain parts of the application. By checking whether a user is logged in, you can ensure that only authenticated users are able to access certain parts of the application. For example, if your application has a private area that is only accessible to logged-in users, you can check the login status of each user before allowing them to access the private area. If a user is not logged in, you can redirect them to a login page or display an error message indicating that they do not have permission to access the private area.
- 7- Outputting success or failure messages using `isset`: It is important to use the `isset` function to check whether a variable has been set when outputting success or failure messages, in order to avoid errors and improve the stability and reliability of the application. The `isset` function is used to determine whether a variable has been set, which means that it has been assigned a value other than `NULL`. If a variable has not been set, attempting to access it can result in an error.

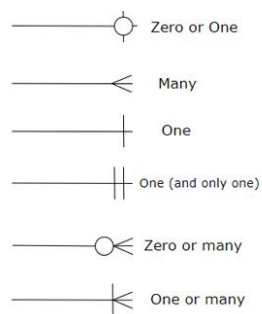
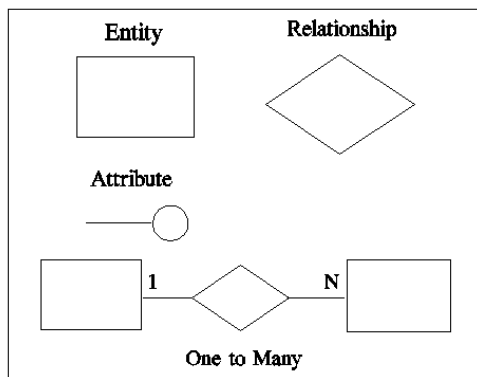
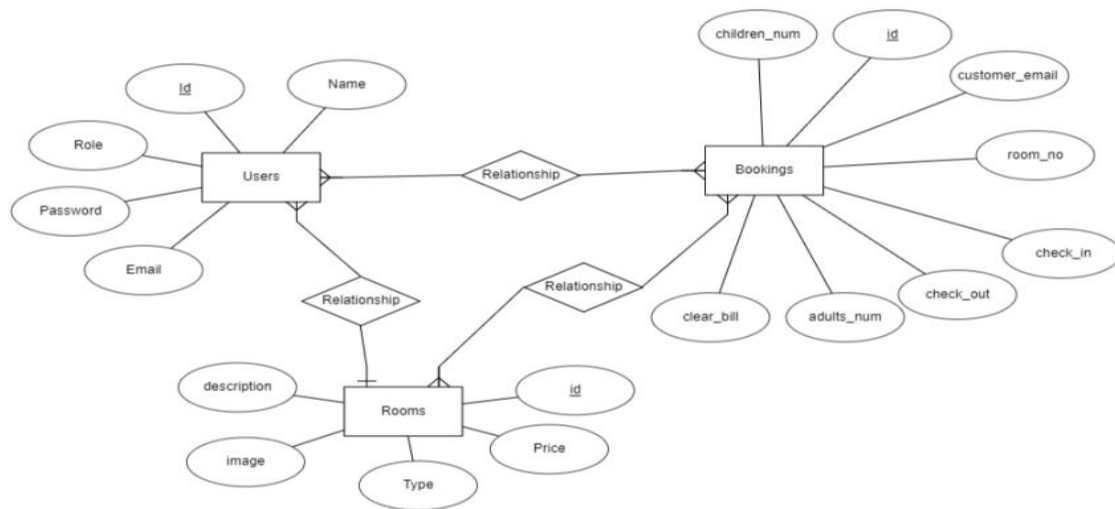
Security features implemented on the logout page:

- 1- Invalidating the session ID: this is a security measure that can help protect against session hijacking attacks. Session hijacking is a type of attack in which an attacker intercepts and uses a valid session ID to gain unauthorized access to a user's account. By invalidating the session ID, the script makes it impossible for an attacker to use the session ID to gain access to the user's session, even if they managed to obtain the session ID. To invalidate a session ID, the script can destroy the session and unset the cookie that contains the session ID. This can be done using the `session_destroy` function in PHP.
- 2- Secure flag: The secure flag is an optional flag that can be used when setting a cookie in a web application. When the secure flag is set, the browser will only send the cookie over an encrypted connection (e.g., HTTPS). This helps to protect the cookie from being intercepted by an attacker, as the attacker would not be able to decrypt the cookie without also intercepting the encrypted connection. Using the secure flag is important for improving the security of web applications, as it helps to protect sensitive information (such as session IDs) from being intercepted by attackers. It is especially important to use the secure flag when transmitting sensitive information, such as login credentials, over the internet.
- 3- HTTPOnly flag: The HTTPOnly flag is an optional flag that can be used when setting a cookie in a web application. When the HTTPOnly flag is set, the browser will only allow the cookie to be accessed through the HTTP protocol, and not through client-side scripts such as JavaScript. This helps to prevent cross-site scripting (XSS) attacks, in which an attacker injects malicious code into a website and gains access to the user's session through a client-side script. Using the HTTPOnly flag is important for improving the security of web applications, as it helps to prevent XSS attacks and protect sensitive information (such as session IDs) from being accessed or manipulated by attackers. It is especially important to use the HTTPOnly flag when transmitting sensitive information, such as login credentials, over the internet.
- 4- Redirecting to the login page: Redirecting the user to the login page after logging them out is a security measure that can help protect against session fixation attacks. Session fixation is a type of attack in which an attacker tries to reuse a valid session ID to gain unauthorized access to a web application. By redirecting the user to the login page after logging them out, the script ensures that the user cannot simply hit the back button in their browser to regain access to the application. This helps to prevent the attacker from being able to reuse the valid session ID to gain access to the application.

Diagrams:

Entity Relationship Diagram

ER diagrams are helpful for developing and managing databases as well as for comprehending a system's data needs. They can be used as the foundation for the physical design of the database and to convey the structure of a database to stakeholders including database administrators, developers, and users.



An entity-relationship (ER) diagram is a visual representation of the connections between the entities in a database. It is used to represent the data needs of a system and illustrate the relationships between the various system components.

A variety of elements make up an ER diagram, including:

Entities: These are rectangles that stand in for the concepts or objects contained in the database. An entity has a collection of attributes, which are the properties or qualities of the entity.

Relationships: These are represented as diamonds and demonstrate the connections between the different entities. One-to-one, one-to-many, or many-to-many relationships are all attainable.

Attributes: These would be represented by ellipses and represent the characteristics or properties of an entity. A data type, such as text, numeric, or date, and a name are both associated with an attribute.

User Use Case Diagram

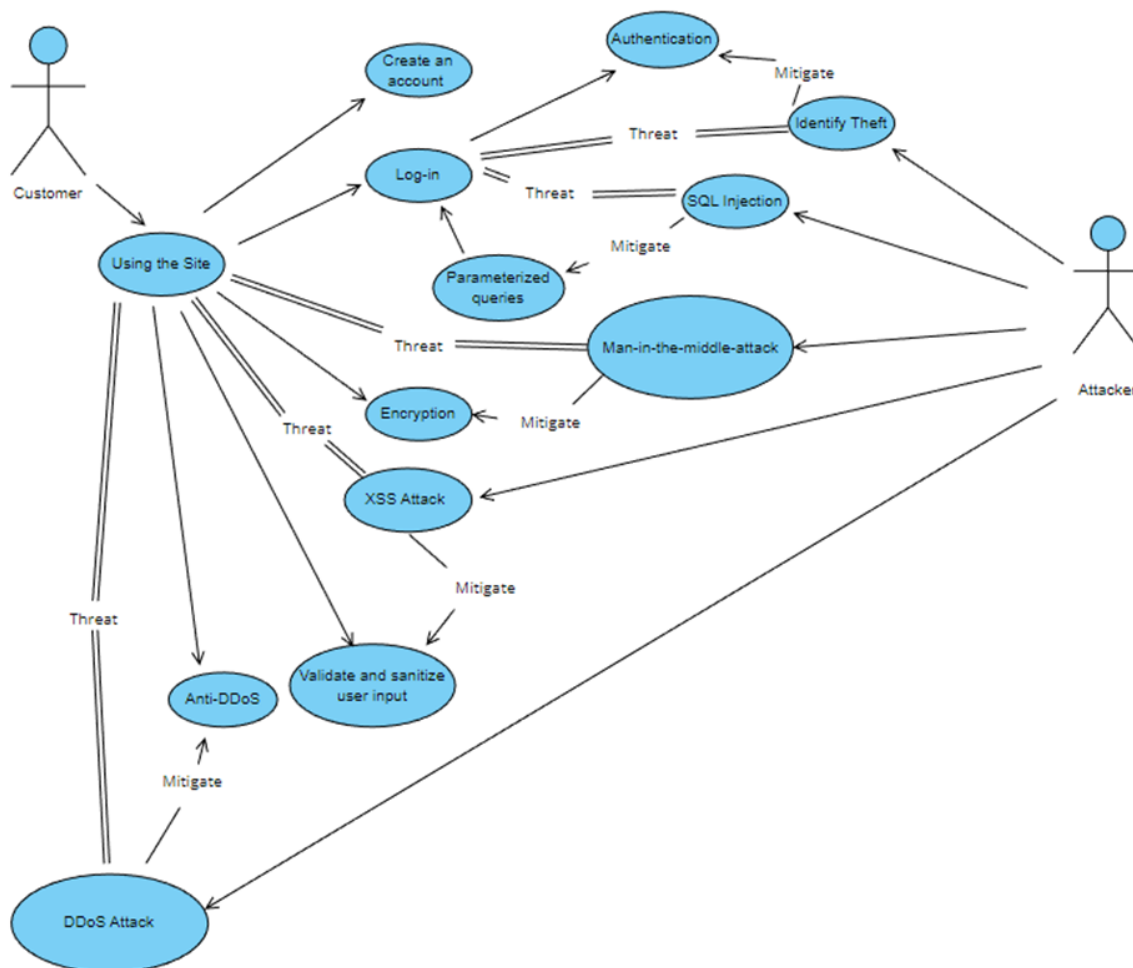
In order to represent the functional requirements of a system, use case diagrams employ graphical representations of interactions between a system's users, or actors, and the system. It is used to provide the responses that the system should provide to an actor's request and the actions that should be taken in response to those requests.



Actors: These stand in for the outside parties that engage with the system. A person, an organization, or another system can all be actors.

Associations: these are lines that connect actors to use cases and imply the interactions between them.

An attacker use case diagram is a visual portrayal of how an attacker interacts with a system with the aim of identifying potential ways to undermine the system's security.



The attacker use case diagram has the same components as a typical use case diagram, including:

Actors: These resemble the external entities that engage with the system, including the attacker and any additional entities that may be involved in the attack.

Use cases: These illustrate the steps an attacker could take to get the system into trouble. Each use case has an oval representation, and its name includes a verb phrase that describes the action.

Associations: these are lines that connect actors to use cases and imply the interactions between them.

Attacker use case diagrams are helpful for identifying and comprehending the potential ways that an attacker can attempt to breach a system's security. They can be used to find possible weak spots and to plan defenses against or attenuate attacks.

GitHub Repository:

<https://github.com/fekk1i/HotelBooking.git>