ABU DHABI POLYTECHNIC

INFORMATION SECURITY ENGINEERING TECHNOLOGY

COURSE PROJECT REPORT

WEB AND DATABASE SECURITY

NCS-2305

Secure E-commerce Platform with Oracle Database:Implementation and Analysis

---

Authors:
Fatima Alhammadi M00005525
Maryam Alnouaimi M00005520
Shamma Alnaqbi M00005441
Rowdah Alhosani M00005440
Falah Alyammahi A00061165

Instructor:
Dr. Nedal ABABNEH

June 19, 2024

Abstract

In modern interlinked computer-based systems, security is of utmost importance. Safeguarding security is becoming increasingly difficult as attack technologies become more sophisticated and accessible. Most security violations in systems occur due to malicious users or code penetrating security barriers, affecting the system by changing its behavior or extracting information. Such actions are identified as intrusions. An intrusion detection system (IDS) collects and analyzes data within a computer or network to identify possible security violations, including both intrusions and misuse. This project focuses on developing and securing a web-based e-commerce platform using Oracle Database, providing practical experience in database server security and administration, enforcing security policies, and mitigating security threats.

I. Introduction

The objective of this project is to develop and secure a web-based e-commerce platform utilizing Oracle Database. The project involves setting up the Oracle database server, implementing robust security policies, managing user authentication and authorization, and safeguarding the system against common vulnerabilities identified by OWASP. Through this project, students will gain practical experience in database server security and administration, develop and enforce security policies, and learn to mitigate security threats effectively.

II. Related Work

When designing and securing a web-based e-commerce platform using Oracle Database, it's essential to consider prior research, existing frameworks, and methodologies related to database security, web application security, and e-commerce platforms. Here are some key areas of related work that provide a foundation for this project:

A. Database Security

1) Oracle Database Security Guidelines:

- **Oracle Security Guide**: Oracle Corporation provides comprehensive documentation on securing Oracle Databases. The guide covers topics such as user authentication, data encryption, auditing, and access control[**?**].
- **Data Encryption**: Studies have shown the importance of encrypting sensitive data both at rest and in transit to prevent unauthorized access and ensure data integrity[**?**].

2) Best Practices in Database Security:

- **Least Privilege Principle**: Implementing the least privilege principle in database security is crucial. It involves granting users the minimum level of access necessary to perform their jobs, thereby reducing the risk of data breaches[**?**].
- **Database Auditing**: Regular auditing and monitoring of database activities can detect and prevent unauthorized access and malicious activities. This includes tracking user activities and access patterns[**?**].

B. Web Application Security

1) OWASP Top Ten Vulnerabilities:

- **SQL Injection**: Research on SQL injection vulnerabilities has shown that using prepared statements and parameterized queries can effectively prevent such attacks[**?**].
- **Cross-Site Scripting (XSS)**: Studies emphasize the importance of input validation and output encoding to prevent XSS attacks, which are common in web applications[**?**].
- **Cross-Site Request Forgery (CSRF)**: Implementing anti-CSRF tokens and validating them on the server side can mitigate CSRF attacks, as highlighted by OWASP guidelines[**?**].

2) Secure Coding Practices:

- **Input Validation and Sanitization**: Ensuring that all user inputs are validated and sanitized to prevent malicious data from being processed by the application[**?**].
- **Content Security Policy (CSP)**: Using CSP to control resources the browser is allowed to load, thereby mitigating XSS and other injection attacks[**?**].

C. E-commerce Platform Security

1) Secure E-commerce Architectures:

- **Designing Secure E-commerce Systems**: Research has been conducted on secure architectures for e-commerce platforms, emphasizing the importance of secure payment processing, user data protection, and transaction security[**?**].
- **PCI-DSS Compliance**: Ensuring that the e-commerce platform complies with Payment Card Industry Data Security Standard (PCI-DSS) is critical for protecting payment information[**?**].

2) Case Studies and Practical Implementations:

- **Real-World Implementations**: Several case studies on securing e-commerce platforms provide insights into practical challenges and solutions. For example, the implementation of two-factor authentication and secure payment gateways in major e-commerce platforms has proven effective in enhancing security[**?**].

D. Security Tools and Frameworks

1) Penetration Testing and Vulnerability Scanning:

- **Tools like OWASP ZAP and Burp Suite**: These tools are widely used for detecting vulnerabilities in web applications and databases. They offer functionalities for automated scanning, manual testing, and reporting[**?**].
- **Metasploitable and DVWA**: These intentionally vulnerable environments are used for training and testing security measures, providing a practical way to understand and mitigate real-world vulnerabilities[**?**].

2) Security Training Platforms:

- **Hack The Box and WebGoat**: Platforms that offer hands-on experience with real-world security challenges, allowing practitioners to hone their skills in a controlled, legal environment[**?**], [**?**].

III. Project Phases and Deliverables (Results-Analysis)

Phase 1: Oracle Database Setup and Basic Security Configuration

A. Install Oracle Database

1) **Prepare the Server:** Ensure your server meets the hardware and software prerequisites specified in the Oracle Database documentation.

2) **Download Oracle Database Software:** Visit the Oracle Technology Network (OTN) website and download the Oracle Database installer appropriate for your operating system.

- Install SQL*Plus and DB Assistant from here.

Figure 11

- Install Oracle Sql Developer from here.

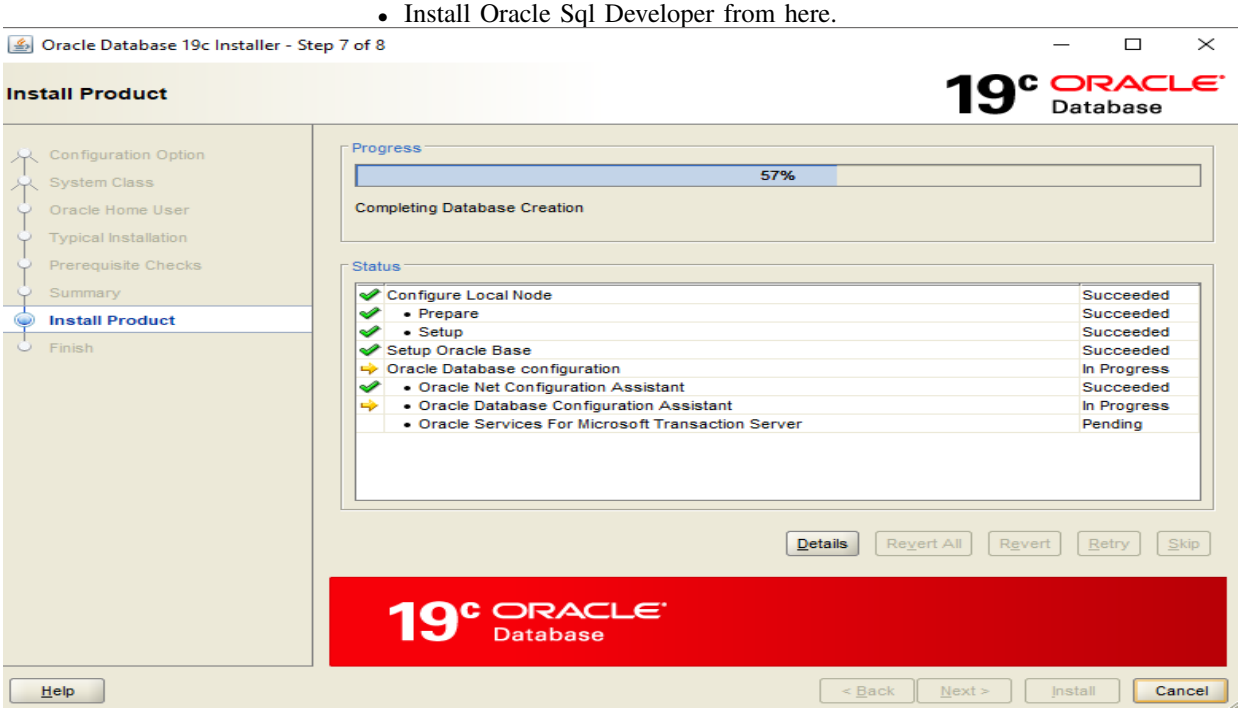


Figure 12

3) **Run the Installer:** Unzip the downloaded installer file and run the installer executable (`runInstaller`).
4) **Complete Installation:** Allow the installer to finish installing the software and configuring the database instance.
5) **Configure Database:** Follow the prompts to configure the database instance, providing details such as database name, Oracle home directory, and administrative credentials.

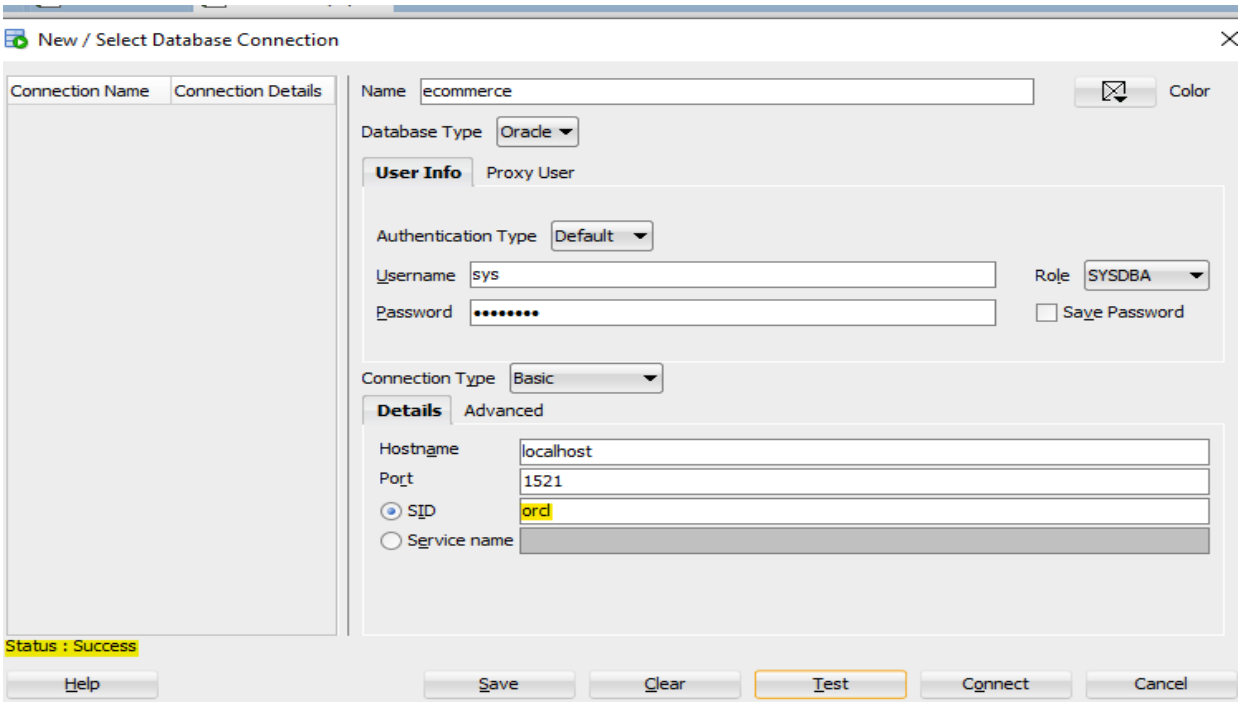


Figure 13

B. *Post-Installation Tasks*

1) **Change Default Ports:** Modify the listener port in `listener.ora` (typically located in the ORACLE-HOME/network/admin directory) to a non-default value.

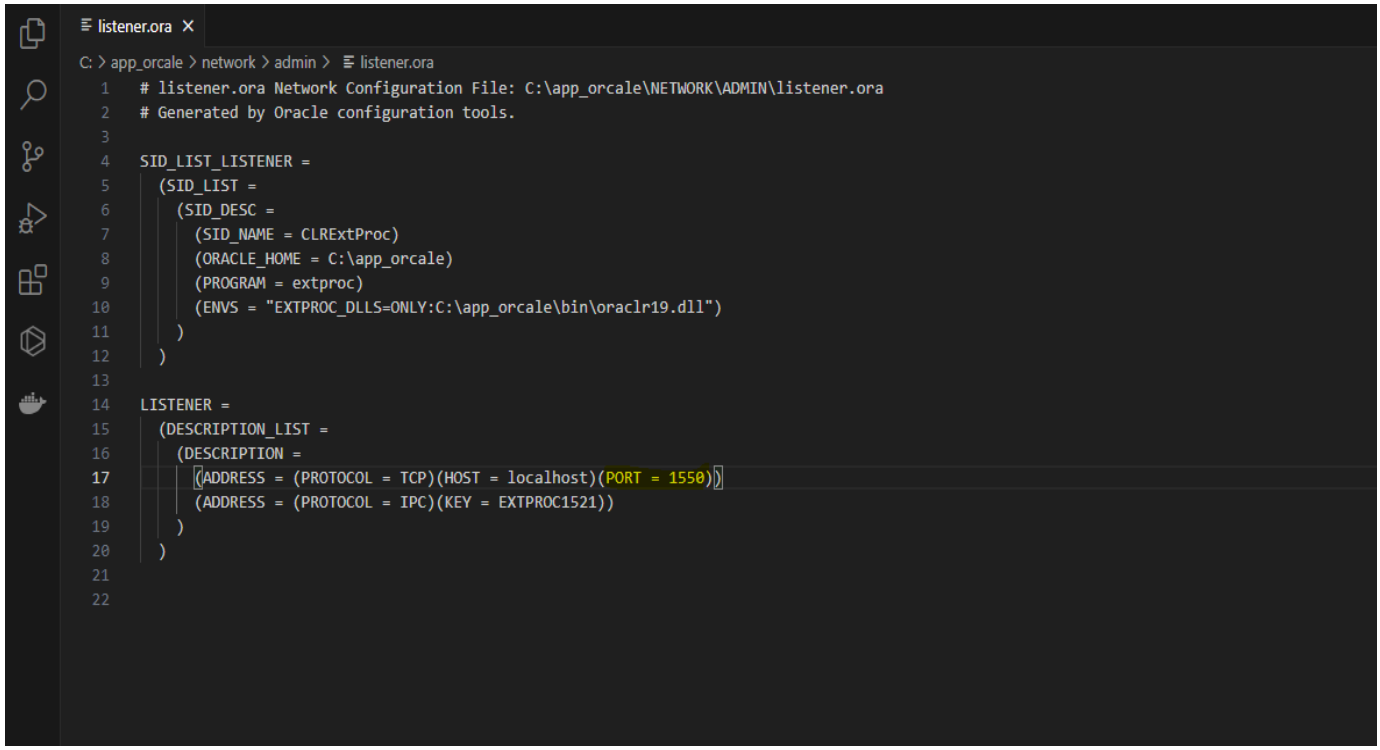


Figure 14

2) **Disable Unused Features:** Review and disable unnecessary Oracle Database features using Oracle Database Configuration Assistant (DBCA).

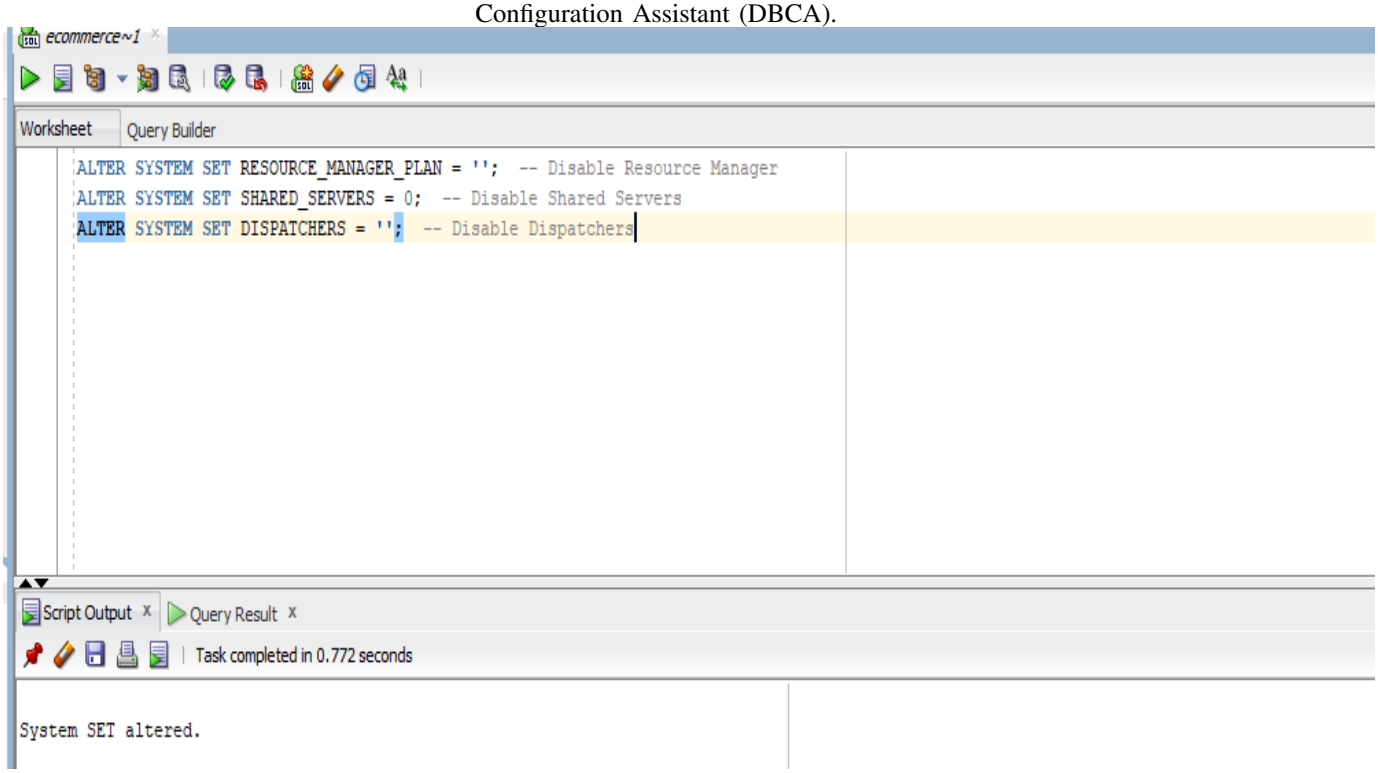


Figure 15

3) **Implement Access Controls:** Set strong passwords for administrative accounts and use roles and profiles to control user access.

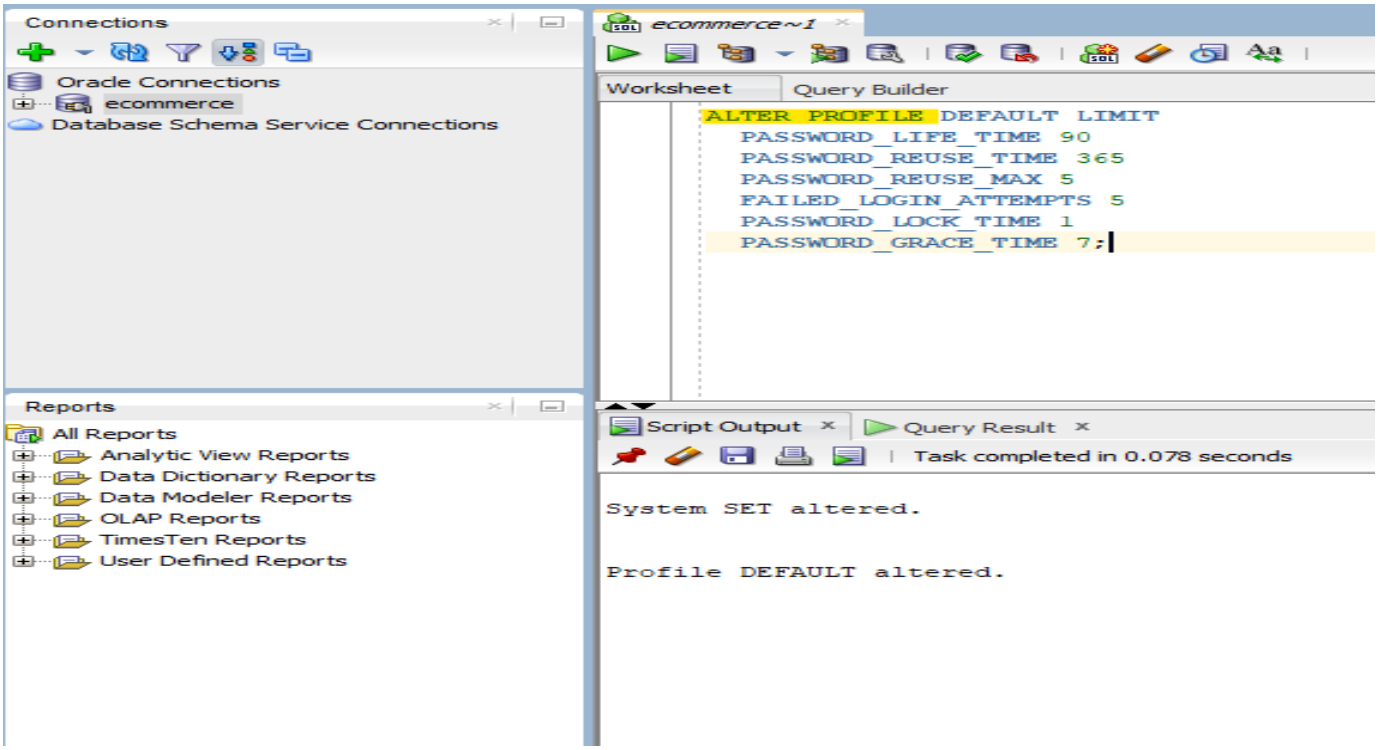


Figure 16

4) **Enable Network Encryption:** Configure Oracle Net Services (SQL*Net) to use encryption for network traffic.
5) **Apply Security Patches:** Regularly apply Oracle Database patches and updates to mitigate security vulnerabilities.

BY: Fatima Alhammadi M00005525

Phase 2: Security Policy Development

C. Password Policies

1) **Password Complexity:** .

- All user passwords must meet the following complexity requirements: Minimum length: 12 characters Must include at least one uppercase letter, one lowercase letter, one number, and one special character

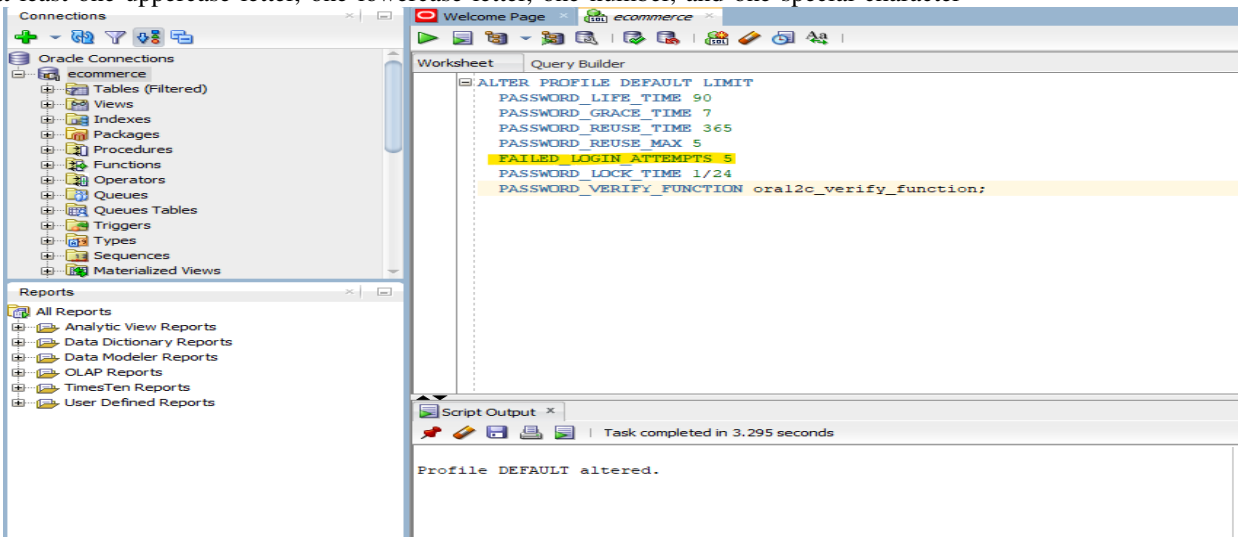


Figure 21

- Passwords must be changed every 90 days.

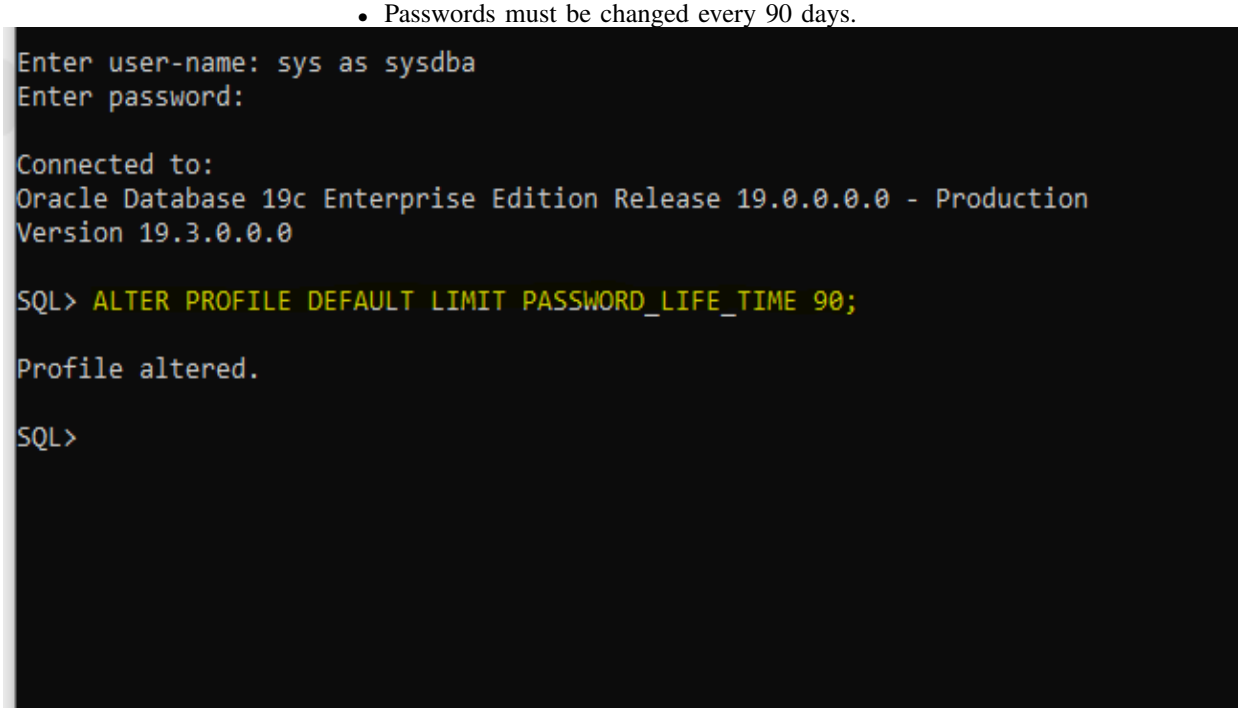


Figure 22

- Accounts will be locked for 1 hour after 5 failed login attempts.

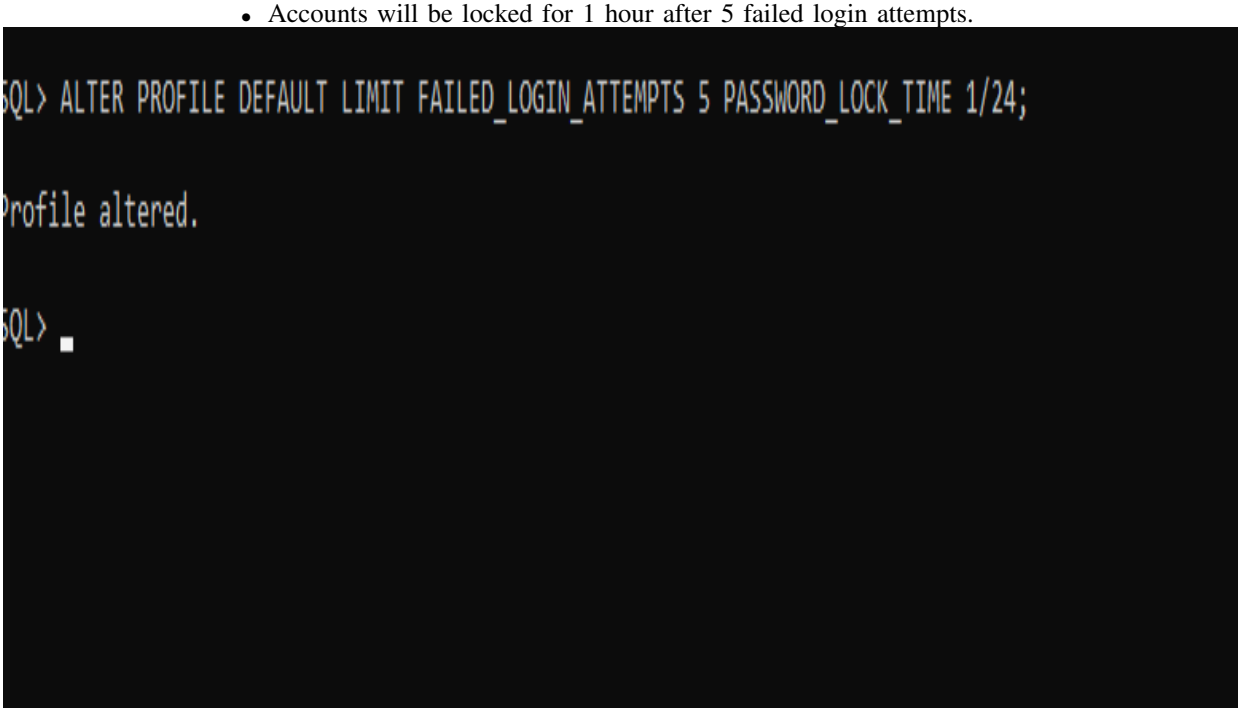


Figure 23

2) **Data Encryption:** All sensitive data must be encrypted at rest using Transparent Data Encryption (TDE), All data transmitted between the database and client applications must be encrypted using Oracle Net Services.

C: > app_orcale > network > admin > sqlnet.ora

```
# sqlnet.ora Network Configuration File: C:\app_orcale\NETWORK\ADMIN\sqlnet.ora
# Generated by Oracle configuration tools.

# This file is actually generated by netca. But if customers choose to
# install "Software Only", this file wont exist and without the native
# authentication, they will not be able to connect to the database on NT.

SQLNET.AUTHENTICATION_SERVICES= (NTS)

SQLNET.ENCRYPTION_SERVER = REQUIRED
SQLNET.ENCRYPTION_TYPES_SERVER = (AES256)


NAMES.DIRECTORY_PATH= (TNSNAMES, EZCONNECT)
```

Figure 24

3) **Backup Procedures:** Perform full database backups weekly and incremental backups daily.

```
rman target /

RMAN> CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;

RMAN> CONFIGURE BACKUP OPTIMIZATION ON;

RMAN> BACKUP DATABASE PLUS ARCHIVELOG;
```

Figure 25
BY:Maryam Alnouaimi M00005520

Phase 3: Authentication and Authorization Configuration

D. User Authentication Mechanisms

1) **Password Authentication:** .

- Use strong password policies (minimum length, complexity requirements).
- Regular password changes and account lockout mechanisms

Potential Issues: *Weak passwords and user resistance.* **Resolutions:***Enforce complex passwords and educate users on strong password practices*

2) **Multi-Factor Authentication (MFA):** Implement MFA using Oracle Advanced Security. Combine passwords with another factor (e.g., OTP, biometric). **Potential Issues:** *Complexity and user inconvenience.* **Resolutions:***Provide clear setup instructions and support.*

a) **MFA Configuration :** .

- Enable Oracle Advanced Security

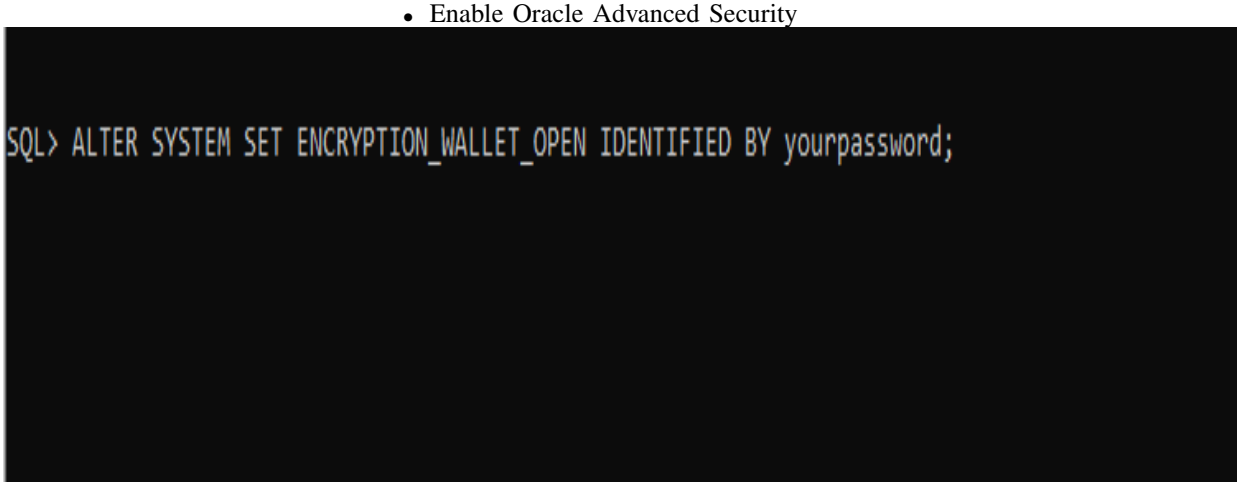


Figure 31

- Configure MFA:

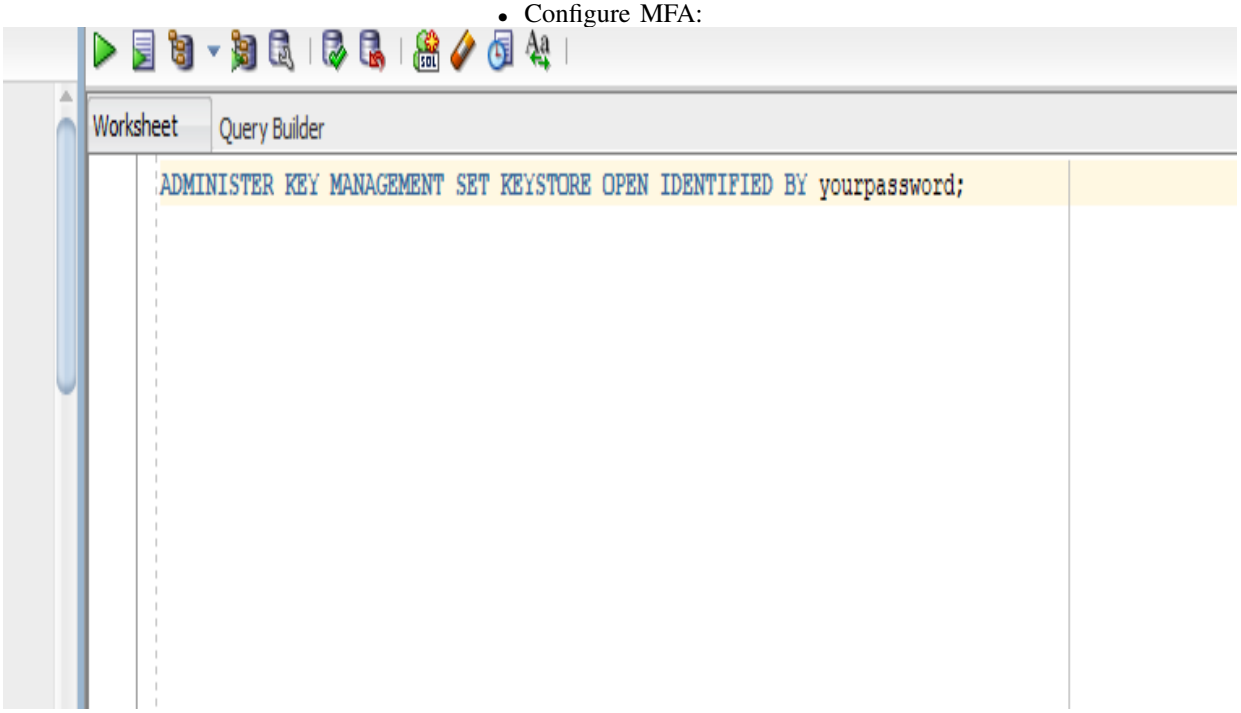


Figure 32

E. User Authorization (Role-Based Access Control (RBAC)): 1)Define roles with specific privileges. 2)Assign users to roles based on their job functions.

Potential Issues: *Over-privileged roles and management complexity.* **Resolutions:***Regularly review and audit roles, apply the least privilege principle.*

a) **RBAC Setup :** .

- Create Roles:

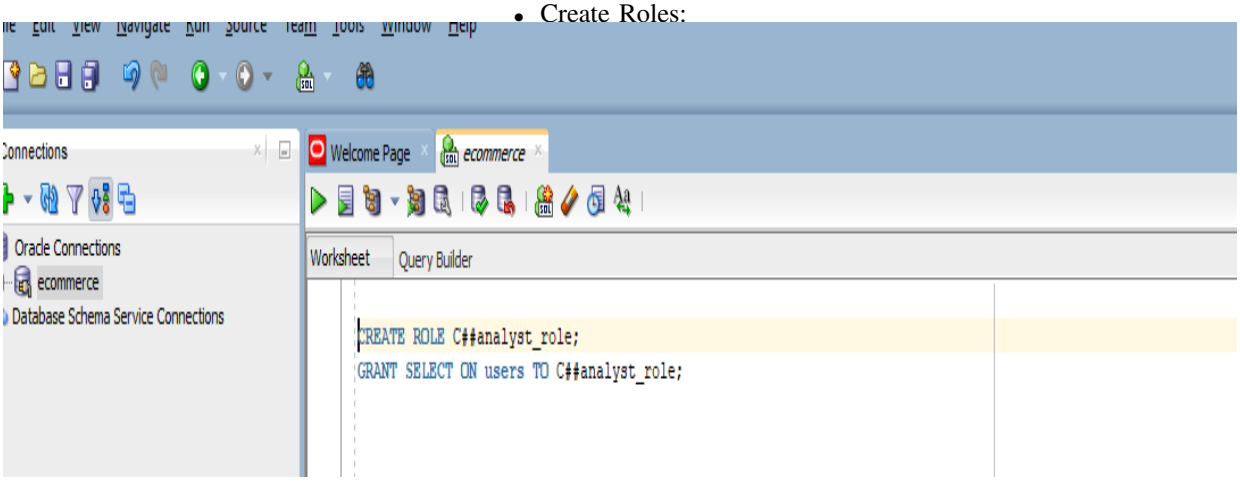


Figure 33

- Assign Roles to Users:

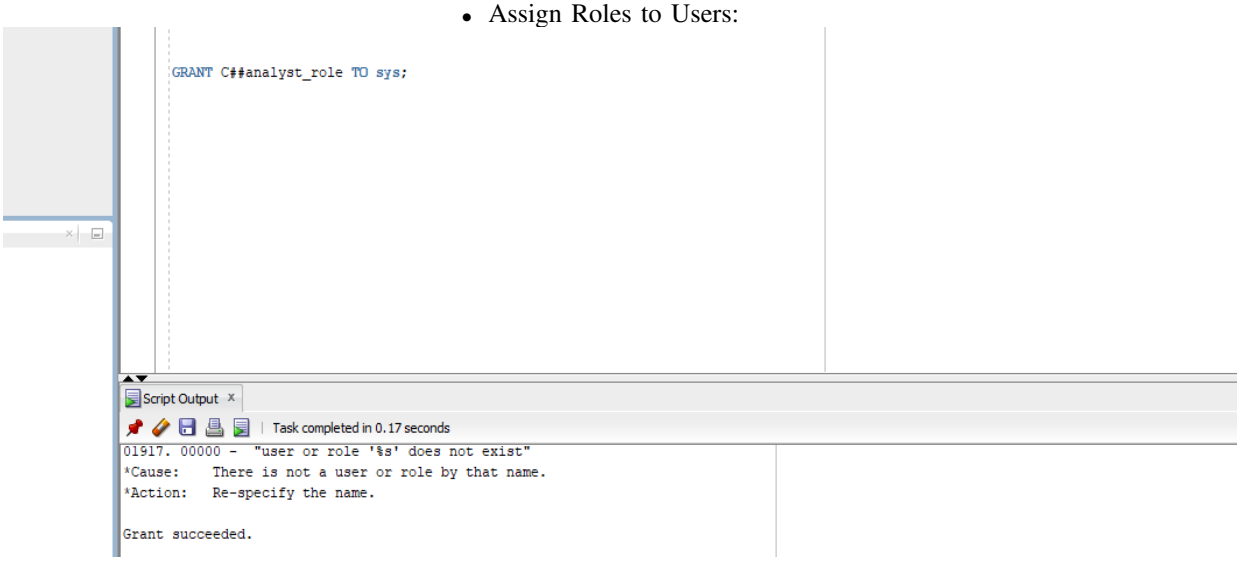


Figure 34

Conclusion

The authentication and authorization configurations significantly enhance Oracle Database security. Addressing potential issues through user education and regular audits ensures robust protection against unauthorized access.

BY:Shamma Alnaqbi M00005441

Phase 4: Access Control Systems Implementation

F. Implement Fine-Grained Access Control Policies

1) **Review Requirements and Existing Policies:** .
   - Understand the current access control requirements.
   - Review any existing policies and identify gaps or areas for improvement.

2) **Identify Sensitive Data and Access Requirements:**
   - Identify sensitive data elements within the databases.
   - Determine who needs access to which data and under what conditions.

3) **Design Access Control Policies:**
   - Define fine-grained access control policies based on the identified requirements.
   - Determine access levels (read, write, execute) for different users or roles.
   - Specify conditions under which access is granted (e.g., time-based, role-based).

4) **Use Database Views:**
   - Utilize stored procedures to encapsulate data access logic.
   - Control access by granting permissions to execute specific procedures rather than direct table access.

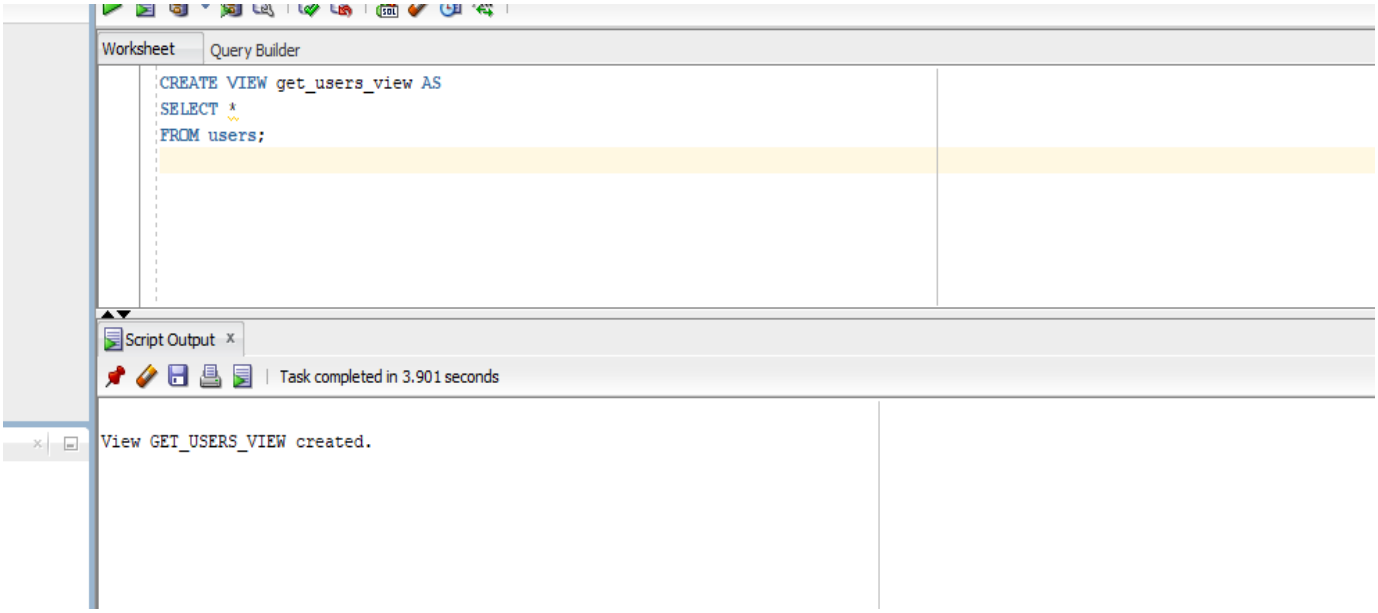


Figure 41

5) **Implement Stored Procedures:**
   - Create database views to limit access to specific columns or rows of data.
   - Views can provide a controlled interface to the underlying tables, masking sensitive data where necessary.

```
CREATE OR REPLACE PROCEDURE get_sensitive_data (IN user_id INT)
LANGUAGE SQL
AS $$
BEGIN
  SELECT sensitive_column1, sensitive_column2
  FROM sensitive_table
  WHERE user_id = user_id;
END;
$$;
```

Figure 42

6) **Grant Permissions:**

- Assign permissions to roles or individual users based on the access control policies defined.
- Use GRANT and REVOKE statements in your database management system (DBMS) to manage permissions

```
GRANT SELECT ON sensitive_data_view TO role_name;

GRANT EXECUTE ON FUNCTION get_sensitive_data TO role_name;
```

Figure 43

Phase 5: Database System Administration Tasks

G. Perform Routine Administration Tasks

1) **Backup:** Ensure that database backups are taken regularly to prevent data loss.
   - Full Database Backup
   - Incremental Backup
   - Archive Log Backup

   Steps:

   - Connect to RMAN (Recovery Manager):

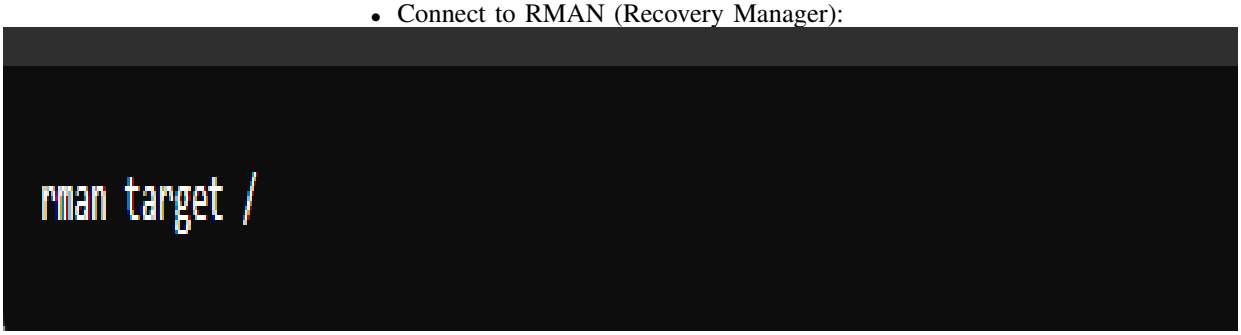


Figure 51

   - Execute the backup command

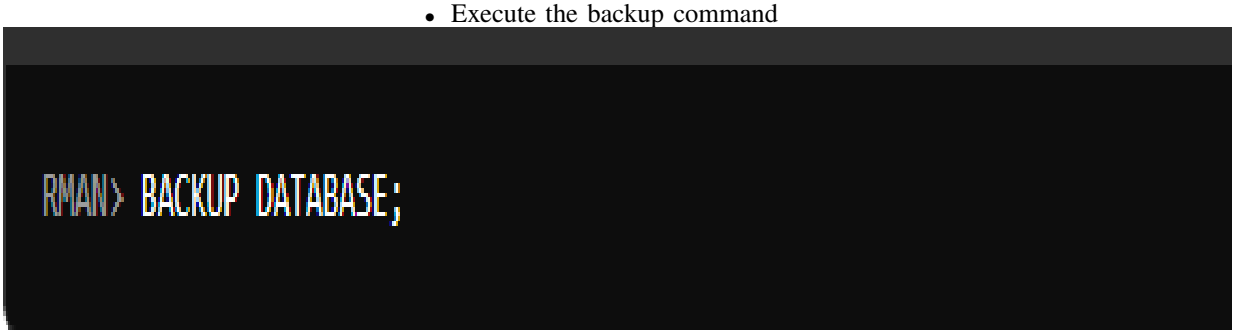


Figure 52

   - Verify the backup

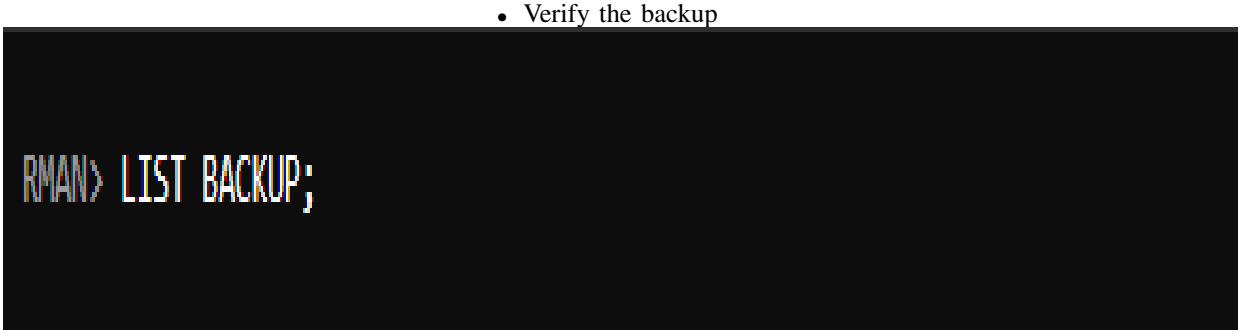


Figure 53

2) **Performance Tuning:** Optimize database performance by monitoring and tuning SQL queries, memory allocation, and storage.
   - SQL Query Optimization
   - Index Management
   - Memory Allocation(SGA,PGA)
   - Storage Management

```sql
-- Identify long-running queries
SELECT sql_id, elapsed_time, sql_text
FROM v$sql
WHERE elapsed_time > 10000000
ORDER BY elapsed_time DESC;

-- Rebuild indexes
ALTER INDEX index_name REBUILD;
```

Figure 54

3) **Monitoring:** Continuously monitor the database to ensure availability, performance, and security.
   - Oracle Enterprise Manager (OEM)
   - Custom SQL Scripts
   - Automated Alerts

```sql
-- Check tablespace usage
SELECT tablespace_name,
       ROUND((used_space/total_space) * 100, 2) AS used_percentage
FROM dba_tablespace_usage_metrics;

-- Monitor session activity
SELECT sid, username, program, status
FROM v$session
WHERE status = 'ACTIVE';
```

Figure 55

H. Automate Administrative Tasks

a) **Automate Backups:** Schedule regular backups to run automatically.

```
#!/bin/bash
rman target / <<EOF
BACKUP DATABASE;
EXIT;
EOF
```

Figure 56

b) **Automate Performance Reports:** Generate and email performance reports regularly.

```
#!/bin/bash
sqlplus / as sysdba <<EOF
@?/rdbms/admin/awrrpt.sql
EOF

# Email the report
mail -s "AWR Report" dba@example.com < /path/to/awr/report.html
```

Figure 57

c) **Automate Monitoring Alerts:** Send alerts for specific conditions (e.g., high CPU usage, low tablespace).

```
#!/bin/bash
sqlplus / as sysdba <<EOF
SET PAGESIZE 0 FEEDBACK OFF VERIFY OFF HEADING OFF ECHO OFF
SPOOL /tmp/tablespace_usage.txt
SELECT tablespace_name,
       ROUND((used_space/total_space) * 100, 2) AS used_percentage
FROM dba_tablespace_usage_metrics
WHERE ROUND((used_space/total_space) * 100, 2) > 80;
SPOOL OFF
EOF

if [ -s /tmp/tablespace_usage.txt ]
then
  mail -s "Tablespace Usage Alert" dba@example.com < /tmp/tablespace_usage.txt
fi
```

Figure 58

********** Logs And Report Summary: **********

```
Date: 2024-06-18
Task: Full Database Backup
Performed by: DBA
Outcome: Successful, backup file created at /u01/app/oracle/backup/backup_20240618.bkp

Date: 2024-06-19
Task: SQL Query Optimization
Performed by: DBA
Outcome: Optimized query with SQL_ID 'abcd1234', reduced execution time from 10s to 2s

Date: 2024-06-20
Task: Tablespace Usage Monitoring
Performed by: Automated Script
Outcome: Sent alert for tablespace 'USERS' exceeding 80% usage
```

Figure 59

```
### Database Administration Report - June 2024

#### Backups
- Full Database Backup performed daily at 2 AM.
- Incremental backups scheduled every 6 hours.
- All backups completed successfully with no errors.

#### Performance Tuning
- Identified and optimized 5 long-running queries.
- Rebuilt 3 indexes to improve query performance.

#### Monitoring
- Automated scripts in place to monitor tablespace usage and session activity.
- Alerts sent for tablespace 'USERS' exceeding 80% usage.

#### Automated Tasks
- Backup scripts scheduled via cron jobs.
- AWR reports generated and emailed weekly.
- Monitoring scripts triggered alerts as expected.

#### Issues and Resolutions
- Temporary performance degradation resolved by increasing SGA size.
- Addressed tablespace usage by adding new data files to 'USERS' tablespace.
```

Figure 5.1.1

BY:Falah Alyammahi A00061165

Phase 6: OWASP Vulnerability Detection and Prevention

I. Vulnerabilities with Orcale

a) **SQL Injection (SQLi):**

An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Criminals may use it to gain unauthorized access to your sensitive data: customer information, personal data, trade secrets, intellectual property, and more. SQL Injection attacks are one of the oldest, most prevalent, and most dangerous web application vulnerabilities. The OWASP organization (Open Web Application Security Project) lists injections in their OWASP Top 10 2017 document as the number one threat to web application security.*SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.*

An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Criminals may use it to gain unauthorized access to your sensitive data: customer information, personal data, trade secrets, intellectual property, and more. SQL Injection attacks are one of the oldest, most prevalent, and most dangerous web application vulnerabilities. The OWASP organization (Open Web Application Security Project) lists injections in their OWASP Top 10 2017 document as the number one threat to web application security.

SQL Injectionl

- *How it Performed:*
  SQL is a query language that was designed to manage data stored in relational databases. You can use it to access, modify, and delete data. Many web applications and websites store all the data in SQL databases. In some cases, you can also use SQL commands to run operating system commands. Therefore, a successful SQL Injection attack can have very serious consequences.To make an SQL Injection attack, an attacker must first find vulnerable user inputs within the web page or web application. A web page or web application that has an SQL Injection vulnerability uses such user input directly in an SQL query. The attacker can create input content. Such content is often called a malicious payload and is the key part of the attack. After the attacker sends this content, malicious SQL commands are executed in the database. SQL is a query language that was designed to manage data stored in relational databases. You can use it to access, modify, and delete data. Many web applications and websites store all the data in SQL databases. In some cases, you can also use SQL commands to run operating system commands. Therefore, a successful SQL Injection attack can have very serious consequences.
- *How to Prevent an SQL Injection:*
  The only sure way to prevent SQL Injection attacks is input validation and parametrized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes. It is also a good idea to turn off the visibility of database errors on your production sites. Database errors can be used with SQL Injection to gain information about your database.
- *Detection:*
  - *Regularly test input fields for SQL injection vulnerabilities using automated tools like SQLMap.*
  - *Manual testing by inputting special characters and SQL keywords into input fields to check for unexpected database responses.*

```
SQL> SELECT * FROM users WHERE username = '' OR 1=1;

        ID USERNAME
---------- ----------------------------------------------------
PASSWORD
--------------------------------------------------------------------------------
         1 testuser
password
```

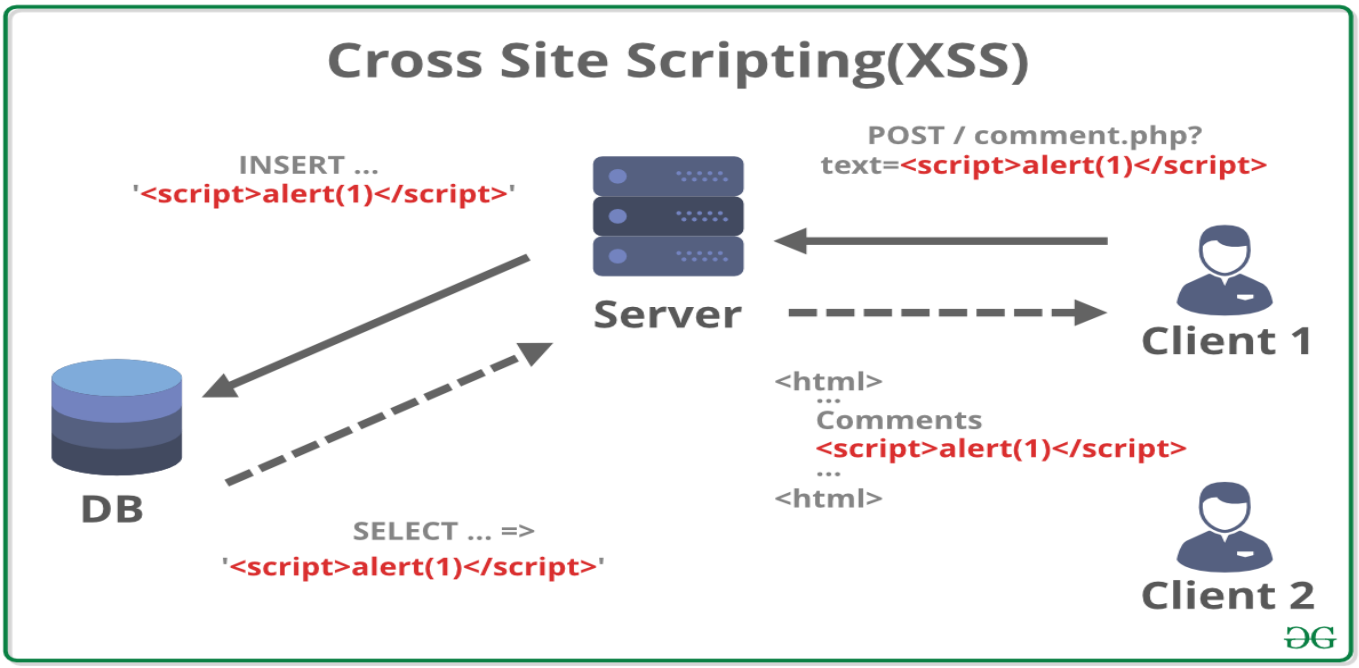
Figure 61

```
-- MySQL, MSSQL, Oracle, PostgreSQL, SQLite
' OR '1'='1' --
' OR '1'='1' /*
-- MySQL
' OR '1'='1' #
-- Access (using null characters)
' OR '1'='1' %00
' OR '1'='1' %16
```

Figure 62 (you can try these)

b) Cross-Site Scripting (XSS):

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.



Cross-Site Scripting (XSS)

- How it Performed:

Unfortunately, there's a slight hitch if you use Chrome. From version 92 onward (July 20th, 2021), cross-origin iframes are prevented from calling alert(). As these are used to construct some of the more advanced XSS attacks, you'll sometimes need to use an alternative PoC payload. In this scenario, we recommend the print() function. If you're interested in learning more about this change and why we like print(), check out our blog post on the subject. As the simulated victim in our labs uses Chrome, we've amended the affected labs so that they can also be solved using print(). We've indicated this in the instructions wherever relevant. *You can confirm most kinds of XSS vulnerability by injecting a payload that causes your own browser to execute some arbitrary JavaScript. It's long been common practice to use the alert() function for this purpose because it's short, harmless, and pretty hard to miss when it's successfully called. In fact, you solve the majority of our XSS labs by invoking alert() in a simulated victim's browser. Unfortunately, there's a slight hitch if you use Chrome. From version 92 onward (July 20th, 2021), cross-origin iframes are prevented from calling alert(). As these are used to construct some of the more advanced XSS attacks, you'll sometimes need to use an alternative PoC payload. In this scenario, we recommend the print() function. If you're interested in learning more about this change and why we like print(), check out our blog post on the subject. As the simulated victim in our labs uses Chrome, we've amended the affected labs so that they can also be solved using print(). We've indicated this in the instructions wherever relevant.*

- How to Prevent an Cross-Site Scripting (XSS):

  The simplest and arguably the easiest form of cross-site scripting vulnerability elimination would be to pass all external data through a filter. Such a filter would remove dangerous keywords, for example, the infamous ¡script¿ tag, JavaScript commands, CSS styles, and other dangerous HTML markups (such as those that contain event handlers.) Many web developers choose to implement their own XSS filter mechanisms. They usually write server-side code (in PHP, ASP, or some other web-enabled development language) to search for keywords and replace them with empty strings. A lot of code uses regular expressions for filtering and replacing. This technique is not a bad one in itself, however, the hackers usually have more experience than web developers and often manage to circumvent simple filters by using techniques such as hex encoding, Unicode character variations, line breaks, and null characters in strings. These techniques must all be catered for and that is why it is recommended to use some sort of library that has been tried and tested by the community at large. Many libraries exist to choose from and your choice will primarily depend on the back-end technology that your web server uses. What is important is that you choose a library that is regularly maintained by a reliable source. XSS techniques keep changing and new ones emerge all the time so your filters will need to be updated periodically to keep abreast with the changing attacks. If you are using Java, then a good place to go to is the OWASP Java Encoder Project. For PHP, there is a comprehensive library called HTML Purifier, which boasts strict standards compliance and better features than other filters. The side effect of filtering techniques is that legitimate text is often removed because it matches forbidden keywords. For example, this article would be incomplete if the Acunetix web server simply filtered out all HTML markup. It would be impossible to include text like ¡script¿ and alert('you have been hacked'). If you want to preserve the original data (and its formatting) as best as possible you need to relax your filters and employ HTML, Script, and CSS escaping techniques.*All XSS attacks affect your web site through some form of client-side user input. Malicious code could come from a simple ¡form¿ submitted by your users or could take a more complex route such as a JSON script, XML web service, or even an exploited cookie. In all cases, the web developer should be aware that the data is coming from an external source and therefore must not be trusted since it may introduce a security vulnerability. The simplest and arguably the easiest form of cross-site scripting vulnerability elimination would be to pass all external data through a filter. Such a filter would remove dangerous keywords, for example, the infamous ¡script¿ tag, JavaScript commands, CSS styles, and other dangerous HTML markups (such as those that contain event handlers.) Many web developers choose to implement their own XSS filter mechanisms. They usually write server-side code (in PHP, ASP, or some other web-enabled development language) to search for keywords and replace them with empty strings. A lot of code uses regular expressions for filtering and replacing. This technique is not a bad one in itself, however, the hackers usually have more experience than web developers and often manage to circumvent simple filters by using techniques such as hex encoding, Unicode character variations, line breaks, and null characters in strings. These techniques must all be catered for and that is why it is recommended to use some sort of library that has been tried and tested by the community at large. Many libraries exist to choose from and your choice will primarily depend on the back-end technology that your web server uses. What is important is that you choose a library that is regularly maintained by a reliable source. XSS techniques keep changing and new ones emerge all the time so your filters will need to be updated periodically to keep abreast with the changing attacks. If you are using Java, then a good place to go to is the OWASP Java Encoder Project. For PHP, there is a comprehensive library called HTML Purifier, which boasts strict standards compliance and better features than other filters. The side effect of filtering techniques is that legitimate text is often removed because it matches forbidden keywords. For example, this article would be incomplete if the Acunetix web server simply filtered out all HTML markup. It would be impossible to include text like ¡script¿ and alert('you have been hacked'). If you want to preserve the original data (and its formatting) as best*

as possible you need to relax your filters and employ HTML, Script, and CSS escaping techniques.

- Detection:
  - Test web applications for XSS vulnerabilities using tools like OWASP ZAP or Burp Suite
  - Manual testing by injecting JavaScript code into input fields and monitoring for execution.

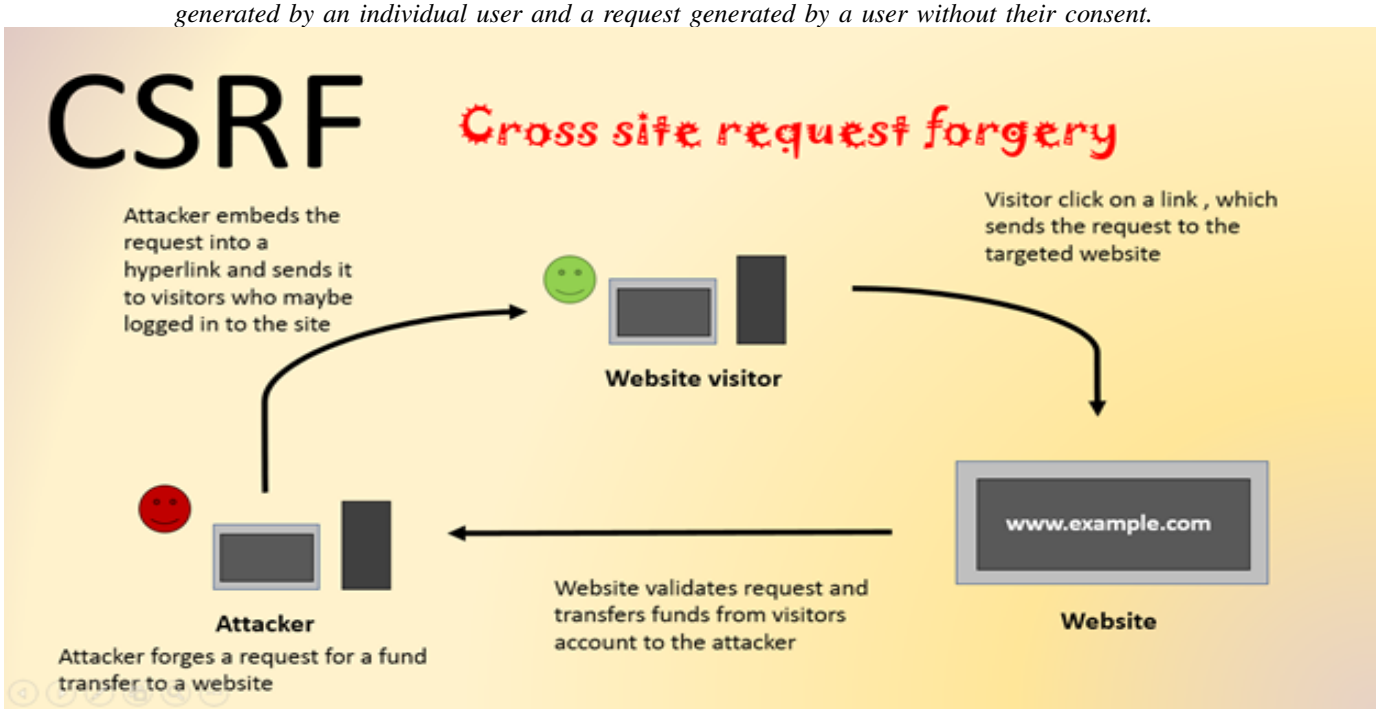
```
-- Input
<script>alert('XSS');</script>


-- Vulnerable Output
<p>Hello, <script>alert('XSS');</script></p>
```

Figure 63

c) Cross-Site Request Forgery (CSRF))

Cross-Site Request Forgery (CSRF) is an attack that forces authenticated users to submit a request to a Web application against which they are currently authenticated. CSRF attacks exploit the trust a Web application has in an authenticated user. (Conversely, cross-site scripting (XSS) attacks exploit the trust a user has in a particular Web application). A CSRF attack exploits a vulnerability in a Web application if it cannot differentiate between a request generated by an individual user and a request generated by a user without their consent.



csrf

- How it Performed:

  Social engineering platforms are often used by attackers to launch a CSRF attack. This tricks the victim into clicking a URL that contains a maliciously crafted, unauthorized request for a particular Web application. The

user's browser then sends this maliciously crafted request to a targeted Web application. The request also includes any credentials related to the particular website (e.g., user session cookies). If the user is in an active session with a targeted Web application, the application treats this new request as an authorized request submitted by the user. Thus, the attacker succeeds in exploiting the Web application's CSRF vulnerability.

- How to Prevent an Cross-Site Request Forgery (CSRF):
  A CSRF secure application assigns a unique CSRF token for every user session. These tokens are inserted within hidden parameters of HTML forms related to critical server-side operations. They are then sent to client browsers. It is the application team's responsibility to identify which server-side operations are sensitive in nature. The CSRF tokens must be a part of the HTML form—not stored in session cookies. The easiest way to add a non-predictable parameter is to use a secure hash function (e.g., SHA-2) to hash the user's session ID. To ensure randomness, the tokens must be generated by a cryptographically secure random number generator. Whenever a user invokes these critical operations, a request generated by the browser must include the associated CSRF token. This will be used by the application server to verify the legitimacy of the end-user request. The application server rejects the request if the CSRF token fails to match the test*To defeat a CSRF attack, applications need a way to determine if the HTTP request is legitimately generated via the application's user interface. The best way to achieve this is through a CSRF token. A CSRF token is a secure random token (e.g., synchronizer token or challenge token) that is used to prevent CSRF attacks. The token needs to be unique per user session and should be of large random value to make it difficult to guess. A CSRF secure application assigns a unique CSRF token for every user session. These tokens are inserted within hidden parameters of HTML forms related to critical server-side operations. They are then sent to client browsers. It is the application team's responsibility to identify which server-side operations are sensitive in nature. The CSRF tokens must be a part of the HTML form—not stored in session cookies. The easiest way to add a non-predictable parameter is to use a secure hash function (e.g., SHA-2) to hash the user's session ID. To ensure randomness, the tokens must be generated by a cryptographically secure random number generator. Whenever a user invokes these critical operations, a request generated by the browser must include the associated CSRF token. This will be used by the application server to verify the legitimacy of the end-user request. The application server rejects the request if the CSRF token fails to match the test*
- Detection:
  * Test forms and state-changing requests for CSRF vulnerabilities.
  * Ensure the presence and proper validation of anti-CSRF tokens.

```
-- Malicious Form
<form method="POST" action="http://ecommerce.com/change-email">
  <input type="hidden" name="email" value="attacker@example.com">
</form>

-- Victim clicks link, executes form unknowingly
<a href="http://malicious.com/csrf.html">Click here</a>
```

Figure 64

Fatima Alhammadi M00005525
Maryam Alnouaimi M00005520
Shamma Alnaqbi M00005441
Rowdah Alhosani M00005440
Falah Alyammahi A00061165

IV. Conclusion

The "Secure E-commerce Platform with Oracle Database: Implementation and Analysis" project provides a comprehensive exploration into securing a web-based e-commerce platform utilizing Oracle Database. Through a phased approach, this project addresses critical areas of database server security and administration, culminating in a secure, robust e-commerce solution.

REFERENCES

[1] Oracle Company (https://www.oracle.com/) Oracle's website offers comprehensive information on its cloud applications and platform services, including enterprise software, cloud infrastructure, and database solution

[2] oracle docs (https://docs.oracle.com/en/database/oracle/oracle-database/19/index.html)The Oracle Database 19c documentation site offers comprehensive resources for managing Oracle Database 19c. Key sections include:

[3] techtarget (https://www.techtarget.com/) Explain methods used in orcale as administrator

[4] academia(https://www.academia.edu/) Express Methods for Optimization Orcale DBM

[5] acunetix (https://www.acunetix.com/) Software company introduce tools for cyber security and techigues for production

[6] Bronfman, S. et al. (2019) Vulnerabilities Slava Bronfman, Tzvika Schneider, Ori Goldberg, Roman Kesler, Alexander Kreines, Tomer Gilad. Bochum: Ruhr-Universitat Bochum.

[7] Sawyer, John F. A. (2001b) Orcale. Pergamon/Elsevier.

[8] OWASP WebGoat (https://www.owasp.org/www-project-webgoat): A deliberately insecure web application designed for learning and practicing web application security techniques. It allows users to explore and exploit vulnerabilities in a safe environment.

[9] P. L. Reid, "Encryption and Data Protection in Oracle Databases," Journal of Information Security, 2020.

[10] Hack The Box (https://www.hackthebox.eu): A popular online platform that offers a range of vulnerable machines and challenges to practice penetration testing skills. It provides a handson learning experience in a controlled and legal environment.

[11] portswigger (https://portswigger.net)PortSwigger is a leading provider of web application security solutions, known for its flagship product, Burp Suite. Burp Suite offers various editions for different needs, including Professional, Enterprise, and Community versions, and is widely used for web vulnerability scanning, penetration testing, and security training