Sudan University of Science & Technology

College of Graduate Studies

Improving a Framework for SQL Attack Detection and Prevention on Web Applications

تحسين إطار عمل لمنع و اكتشاف هجوم الحقن علي تطبيقات الويب

A Thesis submitted in partial Fulfillment of the requirement of master degree in computer science

By:

Esraa Yagoop Abd Alhameed Mohamed Ali

Supervision by:

Dr. Faisal Mohammed Abdallah Ali

January - 2023

الآية

بِسْمِ اللهِ الرَّحْمنِ الرَّحِيمِ

قال تعالى:

﴿ بَدِيعُ السَّمَاوَاتِ وَالْأَرْضِ ۖ وَإِذَا قَضَىٰ أَمْرًا فَإِنَّمَا يَقُولُ لَهُ كُنْ فَيَكُونُ ﴾

﴿ البقرة: 117 ﴾

صدق الله العظيم

ACKNOWLEDGEMENT

I would first like to thanks my God to give me another chance to complete my higher Education and to learn enormous skills and knowledge through all these years of study at the university.

To my **Supervisor Dr. Faisal Mohammed Abdullah Ali**

To my parents for their love and support throughout my life,

To my beloved Sisters, my husband, my children, and everyone who have been a part of my life.

Abstract

The website become are more common used these days according to its features and used to present services, this making these applications more susceptible to be hacked, common hacking of these sites is SQL injection. The core of this study is improving frame aid the developer to how prevent from that hacking. The researcher builds two websites the first is vulnerable and the second applied of security policy within framework and applying hacking on both them. The results show the validate of frame work invented procedure and security policy used in its (User validation and Prepared statement), and the risk analysis phase is more important in framework to determine vulnerabilities. And also find registration path not effect in the security actively but may affect passively. The researcher recommends by use proposed frame work and the policies to prevent hacking.

المستخلص

نلاحظ في الآونة الاخيرة اتساع استخدام مواقع الانترنت وذلك لما فيها من مميزات وبالتالي اصبحت العديد من الخدمات تقدم من خلال تلك المواقع وكذلك عرض المنتجات، مما يجعلها اكثر عرضة للهجمات، واهم هذه الهجمات هجمات الحقن لقواعد البيانات. ويعتبر الهدف الرئيسي لهذا البحث بناء منهجية تساعد المطورين للوقاية من تلك الهجمات. عمل الباحث على بناء موقعين الاول آمن والاخر غير آمن حيث طُبقت سياسات تأمينه على الاول ووضع كلا الموقعين تحت الاختبار. النتائج اوضحت فاعلية المنهجية او اطار العمل لمقاومة هذا النوع من الهجمات وكذلك السياسات التامينة التي استخدمت عملت على الوقاية من تلك الهجمات بخلاف الموقع الغير آمن، كما اثبت من المهم او الضروري تحليل المخاطر لكل موقع وفقاً لنوع الخدمة التي يقدمها وتاكيد الثغرات التي يمكن استغلالها، كما اثبتت الملاحظة ان صفحات التسجيل لا تمثل تهديد فعال وربما تمثل تهديد غير فعال. يوصي الباحث باتباع المنهجية المقترحة لدراسة أمن موقع مع تطبيقات السياسات المناسبة وفقاً للمنهجية والتحليل الكامل للمخاطر.

Table of Contents

List of Figures

List of Table

List of Abbreviation

Syncopation	**WORED**
SQLIA	SQL injection attack
SQL	Structured Query Language
DDOS	Distributed Denial of Service
RDBMS	Relational Database Management System
XSS	Cross Site Scripting
DNS	Domain Name Server
WPSCAN	Word Press Scan
COMMIX	Command injection exploiter
RIA	Rich Internet application
IDS	Intrusion Detection System
DOS	Denial of Service

Chapter One

Introduction

Chapter One

Introduction

1.1 Introduction:

Cyber security has become an important and interesting topic. Poorly designed web applications may include, improper handling of requests, lack of input validation, lack of output validation, poor business-logic design & implementation [1].

In today's digital world, Web applications are being used in numerous ways in recent years to provide online services such as banking, shopping, social networking, etc. These applications operate with sensitive user information and hence there is greater need for assuring their confidentiality, integrity, and availability. Extensive use of websites and web applications has attracted hackers to attack on it using various tricks and   techniques [2]

SQL injection is one of most used attack we face nowadays. In SQL Injection Attack (SQLIA) the attacker can trick the server to obtain illegal authorization and asses the database using SQL queries. This is because the developers of the applications do not know fully about the attacks by SQL injection and its causes. This research paper focused on how to detect and prevent SQL injection attacks on websites and web applications. Of those attacks, a serious role is controlled by SQL injection attacks (SQLIA). SQL injection attack is one amongst the intense dangers to web application accustomed gain unauthorized access to database or to retrieve the confidential data present on the database. A SQL injection attack is done by insertion or "injection" of a SQL query with the input data from the user to the application. A successful SQL injection can read sensitive data from the database, alter database data and perform query such as Insert/Update/Delete and perform administration operations on the database such as shutdown the Database, recover the data present in a file on the Database and perform some commands on the operating system. [3]

SQLA and Cross Site Scripting Attack (XSS) are perhaps a few of the most common attack techniques used by attackers to manipulate or delete the content through inputting unwanted command strings in the coding part.   The attacker attacks in such a way that the code manipulated will look as the original code. [4]

Most of the companies use anti-virus software and other security measures to prevent Cyber Attacks. There is, however, a significant difference in risk mitigation costs. A recent U.S. study found that large enterprises allocate on average 12% of their IT budget on Cyber Security [5].

1.2 Problem Statement:

The weak input filtration and validation of dynamic web applications and using a single detection and prevention technique against SQL injection attacks.

1.3 Research Objectives:

1.3.1 Main objectives:

1. Design a new Framework based on dynamic Analyzer and tester performed well to detect and prevent the SQL injection attacks.
2. To investigate the effect of poor input validation of SQL query to discriminate the parameters used for injection malicious SQL on the security of server database.
3. Improve the filtration level of a user input from real one and a malicious one on dynamic web applications.

1.3.2 Sub objectives:

1. Decrease the response time that is taken by using tools to prevent and detect SQLIA .
2. The proposed solution also needs less modification of the source code of the web application and use minimum resources of the system.

1.4 Research Importance:

i. Many of the web applications use databases as their back-end data store. Although new web programming languages offer new ways of more secure database programming features, still there are many applications that are vulnerable to SQL injection attack.

ii. Because of the nature of this attack which is unauthorized access to the confidential information and inserting or modifying it, this kind of attack is very popular among attackers and again because of the mentioned reason it is vital to make web applications secure against them.

1.5 Research Scope:

Place Scope: The area which we decide to conduct the research into Internet.

Applied Scope: Web Sites

1.6 Research Methodology:

The proposed solution based on dynamic Analyzer and tester by using some tools and methods, it performed well as four phases to detect and block the SQLIA.

1. **Planning phase:**

1.1 Scope and strategy of the assignment is Determine.

1.2 Existing security policies, standards are used for defining the scope.

2. **Discovery phase:**

2.1 Collect as much information as possible about the system including data in the system, user names and even passwords. This is also called as Fingerprinting.

2.2 Design the system by using Asp.net language & Microsoft SQL Server Management Studio database.

2.3 Scan and Probe into the port.

2.4 Check for vulnerabilities of system by using some tools and methods.

3. **Attack phase:**

3.1 Find exploits for various vulnerabilities that is need necessary security privileges to exploit the system, also using some tools and methods that is used in security

4. **Reporting phase:**

4.1 report must contain detailed findings

4.2 Risks of vulnerabilities found and their Impact on business

4.3 Recommendation and solution, if any

1.7 Expected Results (Hypothesis):

Before a web site can be compromised, an attacker needs to find applications that are vulnerable to SQL injection using queries to learn the SQL application methods and its response mechanisms. The attacker has two ways to identify SQL injection vulnerabilities:

i. **Error messages:**

   The attacker constructs the correct SQL syntax based on errors messages propagated from the SQL server via the front-end web application. Using the errors received, the hacker learns the internal SQL database structure and how to attack by injecting SQL queries via the Web application parameters.

ii. **Blindfolded Injection:**

This technique is used by hackers in situations when there are no error messages or response content is returned from the database. In these cases the attacker has limited ability to learn the backend SQL queries in order to balance the SQL injection query. In the lack of database content output.

1.8 Research Structure:

This research is organized as fives chapters

Chapter One: This chapter provides an Introduction of the research topic, Problem Statement, Importance, Hypothesis, Objective, Bounders of Research, Research Methodology, and Thesis Structure.

Chapter Two: This chapter provides a literature Review and Related work

Chapter Three: This chapter provides the methodology

Chapter Four: This chapter provides the implementation, result and Analysis

Chapter Five: This chapter provides recommendations and Conclusion

Chapter Two

Literature Review and Related Works

2.1 Web Application

A web application (aka website) is an application based on the client-server model. The server provides the database access and the business logic. It is hosted on a web server. The client application runs on the client web browser. Web applications are usually written in languages such as Java, C#, and VB.Net, PHP, ColdFusion Markup Language, etc. the database engines used in web applications include MySQL, MS SQL Server, Posture SQL, SQLite ..etc [6] .

SQLIA's take place largely in web applications, thus working of a web application needs to be understood first. Web applications comprise of three tiers. The first tier is on the users side and has a basic browser. The second tier contains a dynamic content generator write in PHP or JSP. Tier three is where the database resides. This is the tier prone to SQL Injection attack **[7]**.

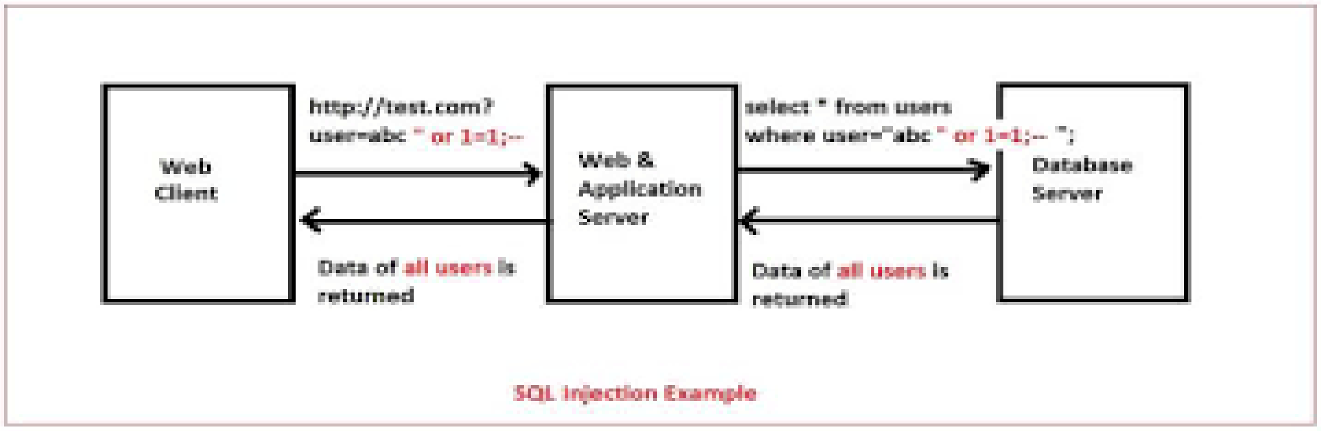


Figure (2.1): Web application and database communication [8]

2.2 Web Application Threats:

Most web applications are hosted on public servers accessible via the Internet. This makes them vulnerable to attacks due to easy accessibility. The following are common web application threats:

2.2 .1 SQL Injection : the goal of this threat could be to bypass login algorithms, sabotage the data, etc.

2.2.2 Denial of Service Attacks : the goal of this threat could be to deny legitimate users access to the resource

2.2.3 Cross Site Scripting (XSS): the goal of this threat could be to inject code that can be executed on the client side browser.

2.2.4 Cookie/Session Poisoning: the goal of this threat is to modify cookies/session data by an attacker to gain unauthorized access.

2.2.5 Form Tampering: the goal of this threat is to modify form data such as prices in e-commerce applications so that the attacker can get items at reduced prices.

2.2.6 Code Injection: the goal of this threat is to inject code such as PHP, Python, etc. that can be executed on the server. The code can install backdoors, reveal sensitive information, etc.

2.2.7 Defacement: the goal of this threat is to modify the page been displayed on a website and redirecting all page requests to a single page that contains the attacker's message [9]

2.3 Countermeasures:

An organization can adopt the following policy to protect itself against web server attacks:

2.3.1 SQL Injection:

Sanitizing and Validating user parameters before submitting them to the database for processing can help reduce the chances of been attacked via SQL Injection. Database engines such as MS SQL Server, MySQL, etc. support parameters, and prepared statements. They are much safer than traditional SQL statements

2.3.2 Denial of Service Attacks :

Firewalls can be used to drop traffic from suspicious IP address if the attack is a simple DOS. Proper configuration of networks and Intrusion Detection System can also help reduce the chances of a DOS attack been successful.

2.3.3 Cross Site Scripting:

Validating and sanitizing headers, parameters passed via the URL, form parameters and hidden values can help reduce XSS attacks.

2.3.4 QS Cookie/Session Poisoning:

This can be prevented by encrypting the contents of the cookies, timing out the cookies after some time, associating the cookies with the client IP address that was used to create them.

2.3.5 Form Tempering:

This can be prevented by validating and verifying the user input before processing it.

2.3.6 Code Injection:

This can be prevented by treating all parameters as data rather than executable code. Sanitization and Validation can be used to implement this.

2.3.7 Defacement:

A good web application development security policy should ensure that it seals the commonly used vulnerabilities to access the web server. This can be a proper configuration of the operating system, web server software, and best security practices when developing web applications [10]

2.4 Web application exploitation:

Malicious users must have to find an input field that lies in the SQL query of the database. In order for an SQL injection attack to take place, the vulnerable website needs direct user input in the SQL query that is injected in the SQL statement. In this way the malicious users inject a payload that is included in SQL query and hence it should be used to attack against the database server. Before the actual attack, first check how the server responds to user's input for authentication mechanism **[**11**]**

2.4.1 Example of authentication credentials:

This was an example of how the user's authentication credentials are checked or verified by the database server.

```
// define POST variables

$Uname  =  $_POST['name'];

$Upassword  =  $_POST['password'];

// sql  query vulnerable to  SQLi

$sql = "SELECT id from users where username = 'Uname' && password = 'Upassword' ";

// execute the  Sql  query by database

Database .execute ($ Sql);
```

As shown the above code is vulnerable to SQL injection, the malicious user can gain access to the web application by submitting the malicious payload in the SQL query that would alter the SQL statement being executed by the database server.

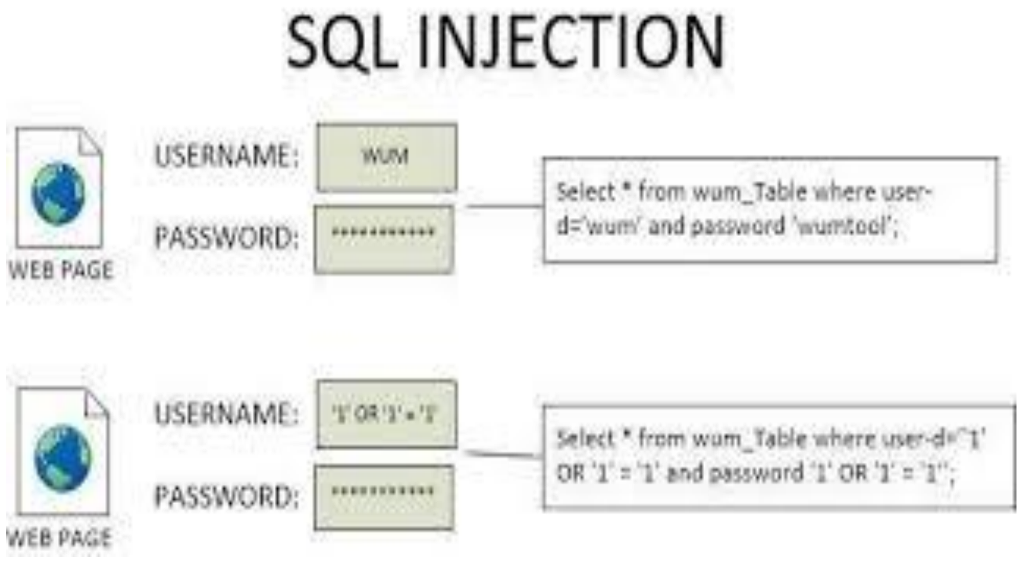
A simple example of an SQL injection payload could be something as simple as setting the password field to password' OR '1'='1′

where this condition is always true.

This would result in the following SQL query being run against the database server.

SELECT id FROM users WHERE username='username' AND password='password' OR 1=1'

Actually in Figure Blow:



Select * from wum_Table where user-d='wum' and password 'wumtool';

Select * from wum_Table where user-d='1' OR '1' = '1' and password '1' OR '1' = '1';

Figure (2.2): Example of SQLIA [12]

2.5 SQL Injection:

According to Wikipedia, SQL injection is a code injection technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed. It is an instance of a more general class of vulnerabilities that can occur whenever one programming or scripting language is embedded inside another. SQL injection attacks are also known as SQL insertion attacks. SQL (Structured Query Language) is a programming language used to work with the databases, commonly Relational Database Management Systems are brought into work. SQL language can be used to update, modify or delete the databases or the tables, columns, rows within RDBMS databases. It is a powerful language to attack databases itself. Attackers can use this language to exploit databases of the web applications and take control of the application without the Administrator's consent. SQL Injection is a malicious attack where malicious users can inject SQL commands (commonly referred to as malicious payload) in SQL statement that controls the web application database (commonly referred to as Relational Database Management System – RDBMS), within the web input field. **[**13**]**

The SQL injection vulnerability can damage any website or web application that is currently using SQL-based database. This is one of the most dangerous web Attacks where the malicious users tend to exploit the web applications. The malicious users can get unauthorized access to the web application, by making use of the SQL injection to bypass the authentication and authorization mechanism defined by any web application. This vulnerability in web applications give illegal access to the malicious user to modify, update or delete the database or make changes to any particular row and columns, in result the data integrity of SQL-based database should be affected.

Keeping the above in mind, when considering the following, it's easier to understand how lucrative a successful SQL injection attack can be for an attacker. An attacker can use SQL injection to bypass authentication or even impersonate specific users. SQL is used to delete records from a database. So an attacker could use an SQL injection vulnerability to delete data from a database. Even if an appropriate backup strategy is employed, deletion of data could affect an application's availability until the database is restored. One of SQL's primary functions is to select data based on a query and output the result of that query.

An SQL injection vulnerability could allow the complete disclosure of data residing on a database server. Since web applications use SQL to alter data within a database, an attacker could use SQL injection to alter data stored in a database. Altering data affects data integrity and could cause repudiation issues, for instance, issues such as voiding transactions, altering balances and other records. Some database servers are configured (intentional or otherwise) to allow arbitrary execution of operating system commands on the database server [14]. Given the right conditions, an attacker could use SQL injection as the initial vector in an attack of an internal network that sits behind a firewall. Show figure blow:

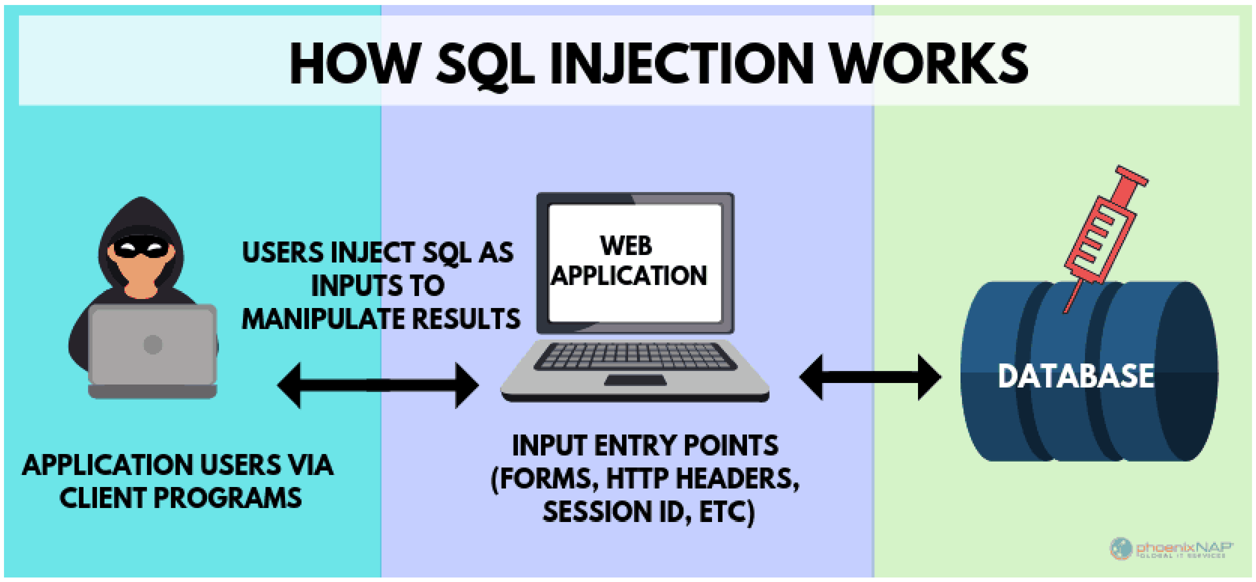

HOW SQL INJECTION WORKS
USERS INJECT SQL AS INPUTS TO MANIPULATE RESULTS
WEB APPLICATION
DATABASE
APPLICATION USERS VIA CLIENT PROGRAMS
INPUT ENTRY POINTS (FORMS, HTTP HEADERS, SESSION ID, ETC)
phoenixNAP

Figure (2.3) : SQL injection and how to prevent an attack[15]

2.6 SQL Injection Attack:

SQL injection is harmful and the risks associated with it provide motivation for attackers to attack the database. The main consequences of these vulnerabilities are attacks on the following:

2.6.1 Authorization:

Critical data that are stored in a vulnerable SQL database may be altered by a successful SQLIA.

2.6.2 Authentication:

If there is no any proper control on input fields inside the authentication page, it may

be possible to login into a system as a normal user without knowing the authenticated user.

2.6.3 Confidentially:

Usually databases are consisting of sensitive data such as personal information, credit card numbers and/ or social numbers. Therefore, loss of confidentially is a terrible problem with SQL Injection vulnerability.

2.6.4 Integrity:

By a successful SQLIA not only an attacker reads sensitive information, but also, it is possible to change or delete this private information.

2.6.5 Database Fingerprinting:

The attacker can determine the type of database being used in backend so that he can use database-specific attacks that correspond to weakness in a particular database management system . **[16]**

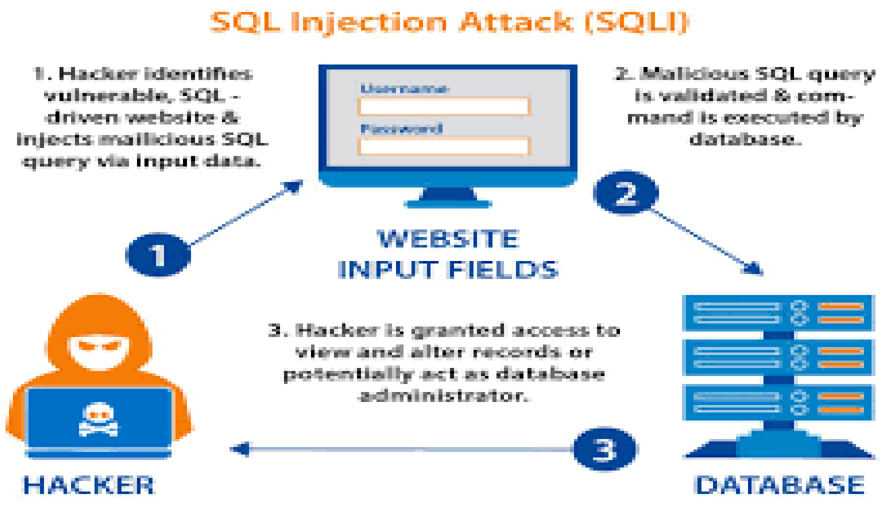

SQL Injection Attack (SQLI)
1. Hacker identifies vulnerable, SQL -driven website & injects mailicious SQL query via input data.
Username
Password
2. Malicious SQL query is validated & command is executed by database.
1
2
WEBSITE INPUT FIELDS
3. Hacker is granted access to view and alter records or potentially act as database administrator.
3
HACKER
DATABASE

Figure (2.4): SQL injection attack-web based Application Security [17]

2.7 Classification of SQLIA:

There are numerous research papers that present a classical SQL Injection attacks, and Detection and Prevention techniques, while the modern SQL Injection attacks, which are the most dangers, are not presented. Therefore, in the following subsections we will discuss and analysis the modern types of SQLIA besides present the classical types.

2.7.1 Classical Types of SQLIA:

As presented in many papers, the classical types of SQLIA can be listed as follows:

1. **Tautologies:** Tautology-based attacks work through injecting code by one or more conditional SQL statement queries in order to make the SQL command evaluate as a true condition such as (1=1) or (- -). The most common use of this technique is to bypass authentication on web pages resulting in access to the database

select* from employees where employee_ID='1' OR '1=1'- '

AND employee_password= '1234';

2. **Piggy-backed Query:** Piggy-backed queries is a type of attack that compromises a database using a query delimiter.

In this method the first query is original whereas the subsequent queries are injected. This attack is very dangerous; attacker can use it to inject virtually any type of SQL command. Piggy-backed query attack:

Select pass from userTable where user _ID=' user'

AND password= 0;

Drop userTable

3. **Logically Incorrect**: Logically Incorrect attack takes advantage of the error messages that are returned by the database for an incorrect query. These database error

messages often contain useful information that allow attacker to find out the vulnerable parameter in an application and the database schema. The SQL query mention Logically Incorrect attack as blow:

Select * from userTable where user _ID=' 1111'

AND password= ' 1234' AND CONVERT(char,no)

4. **Union query:** Union query injection is called as statement injection attack. In this attack, attacker insert additional statement into the original SQL statement. This attack can be done by inserting either a UNION query or a statement of the form ";< SQL statement >" into vulnerable parameter . The result of this attack is that the database returns a dataset that is the union of the results of the original query with the results of the injected query

Select * from userTable where user _ID=' 1111' UNION select8 from memberTable where member _ID=' admin'— '

5. **Stored Procedure**: In this technique, attacker focuses on the stored procedures which are present in the database system. Stored procedures run directly by the database engine. it is a piece of code which is exploitable. Stored procedure gives true or false values for the authorized or unauthorized clients. For SQLIA, attacker will write "; SHUTDOWN; --" with login or secret key.

Select Username from userTable where user _name='user1' AND pass=' ';

6. **Inference:** Using Inference attack, enable the attacker changing the behavior of a database or application. This type of attack can be classified into two well-known techniques, which are: Blind injection and timing attack.

Select pass from userTable where user name='user' and 1=0—AND pass-AND pin=0

Select info from userTable where user name='user' and=1—AND pass= AND pass=0

7. **Alternate Encodings:** This type of attack occurs when attacker modify the injection query via using alternate encoding, such as hexadecimal, ASCII, and Unicode. By means of this way, the attacker can escape from developers' filter, which scan input queries for special known "bad character". When this type of attack combines with other attacks techniques it could be strong, because it can target different layers in the application so developers need to be familiar to all of them to provide an effective defensive coding to prevent the alternate encoding attacks. [18]

Select accounts from userTable where login=”AND pin=0;

Exec(char(0x7687574646f776e))

2.7.2 Modern types of SQLIA

In this subsection is presented an overview for modern types of SQLIA and mention example for each type, as follows:

1. **Fast Flux SQL Injection Attack:**

By using this type of SQLIA , attacker intents to extract data from the database and phishing. The phishing is a social engineering attack in which an attacker fraudulently acquires sensitive information from the user by incorporating a third party.

Traditional phishing host can be detected very easily just by tracking down the public Domain Name Server (DNS) or the IP address. This trace back technique could lead to the shutdown of the hosting websites. The attacker realized that conducting like that attack could have significant effect on load balancing of a server; Therefore , to protect its criminal assets, the operator of phishing websites started using Fast Flux technique. Fast Flux is a Domain Name Server technique which is used to hide phishing and malware distribution sites behind an ever changing network of compromised host.

2. **Compounded SQL Injection Attack:**

This type of SQLIA is a combination derived from a conjunction of SQLIA and other Web Application Attacks. The attacker intents to attack the database and cause more serious effect then other classical types of SQLIA (which are previously discussed).

The rapid development of prevention and detection techniques against various SQLA , enforced the malicious attackers to develop a technique called compounded SQL Injection.

3. **SQL Injection + D Dos (Distributed Denial of Service) Attacks:**

This attack is used to hang a server, exhaust the resources so that the user cannot able to access it. The different SQL commands which can be used in SQL Injection in order to keep track with DDOS Attack is to encode, compress, join etc.

4. **SQL Injection + DNS (Domain Name Service) Hijacking:**

By using this type of attack, the attacker intent to embed the SQL query in DNS request and to capture it and makes its way onto the internet.

5. **SQL Injection + XSS (Cross Site Scripting):**

XSS is the client side code injection attack where an attacker can inject malicious code into the input fields of an application. After inserted the XSS script, it will execute and try to connect with the database of an application. The extraction code for data from the database can be implemented using the I frame command.

SQLIA using Cross Domain Policies of Rich Internet application (RIA): Cross Domain policies is an XML file which gives permission to web client application, such as Java, Adobe Flash, Adobe Reader, etc., to access data in multiple domains.

Cross-domain policies define the list of RIP hosting domains that are allowed to retrieve content from the content providers' domain.

6. **SQL Injection + Insufficient Authentication:**

This type of attack occurs when the user or an administrator is a novice. The security parameters have not been initialized and the attacker can access to the sensitive content without verifying the identity of the user. There by the attacker exploits this vulnerability to inject SQL injection code. Hence, this type of attack is trouble-free as compared with the other types of attack.

The first step is to find whether the application has insufficient authentication and if it has then the SQL Injection attack can take place [19]

2.8 Prevention methods Of SQL Injection Attacks:

There are numbers of methods to prevent the SQL injection attacks and to keep your database safe you can apply some of them, which are:

2.8 .1 Using Prepared Statements (with Parameterized Queries) :

Using Prepared Statements is one of the best ways to prevent SQL injection. It's also simple to write and easier to understand than dynamic SQL queries. This is where the SQL Command uses a parameter instead of inserting the values directly into the command, thus prevent the backend from running malicious queries that are harmful to the database. So if the user entered 12345 or 1=1 as the input, the parameterized query would search in the table for a match with the entire string 12345 or 1=1.

Language specific recommendations:

Java EE – use Prepared Statement () with bind variables

.NET – use parameterized queries like SQL command () or DB command () with bind variables

PHP – use PDO with strongly typed parameterized queries

Hibernate - use create Query () with bind variables (called named parameters in Hibernate)

SQLite - use sqlite3 prepare () to create a statement object

For example, using prepared statement in PHP:

```
$ stmt = $ dbh->prepare('SELECT * FROM customers WHERE  ssn = : ssn');

$ stmt-> bindParam(':ssn' => $ ssn);
```

2.8.2 Using Stored Procedures

Stored Procedures adds an extra security layer to your database be side using Prepared Statements. It performs the escaping required so that the app treats input as data to be operated on rather than SQL code to be executed. The difference between prepared statements and stored procedures is that the SQL code for a stored procedure is written and stored in the database server, and then called from the web app. If user access to the database is only ever permitted via stored procedures, permission for users to directly access data doesn’t need to be explicitly granted on any database table. This way, your database is still safe.

2.8 .3 Validating user input

Even when you are using Prepared Statements, you should do an input validation first to make sure the value is of the accepted type, length, format, etc. Only the input which passed the validation can be processed to the database. It’s like checking who is at the door of your house before you open it and let them in. But remember, this method can only stop the most trivial attacks; it does not fix the underlying vulnerability.

2.8 .4 Limiting privileges

Don’t connect to your database using an account with root access unless required because the attackers might have access to the entire system. Therefore, it’s best to use an account with limited privileges to limit the scope of damages in case of SQL Injection.

2.8 .5 Hiding info from the error message

Error messages are useful for attackers to learn more about your database architecture, so be sure that you show only the necessary information. It’s better to show a generic error message telling something goes wrong and encourage users to contact the technical support team in case the problem persists.

2.8 .6 Updating your system

SQL injection vulnerability is a frequent programming error and it’s discovered regularly, so it’s vital to apply patches and updates your system to the most up-to-date version as you can, especially for your SQL Server.

2.8 .7 Keeping database credentials separate and encrypted

If you are considering where to store your database credentials, also consider how much damaging it can be if it falls into the wrong hands. So always store your database credentials in a separate file and encrypt it securely to make sure that the attackers can’t benefit much. Also, don’t store sensitive data if you don’t need it and delete information when it’s no longer in use.

2.8 .8 Disabling shell and any other functionality you don’t need

Shell access could be very useful indeed for a hacker. That’s why you should turn it off if possible. Remove or disable all functionalities that you don’t need too [20]

2.9 SQLIAs detection and prevention techniques:

Most of methods showed that insufficient input validation detection technique adopted will allow malicious code to be executed without proper verification of its intention.

Therefore, any detection technique should prevent attacker to take great advantages of insufficient input validation which threat the target database and help the attacker to utilize malicious SQL code to conduct attacks easily [21]

SQLIAs detection and prevention techniques have followed various aspects in order to come up with an appropriate solution so as to prevent SQLIAs from being applied to different types of databases. Some of these aspects are :

2.9.1 Static Analysis:

Static analysis is a principle that depends on finding the weaknesses and malicious codes in the system source code prior to reaching the execution stage.

Generally, this principle has been one of the most widely used to detect or prevent SQLIAs.

2.9.2 Runtime Analysis:

It is a technique which has been used to detect a specific type of attacks that should be identified in advance without the need of Modifying the development lifecycle nor the need of the source code of the system. Such a technique depends on tracking the events of the system through its execution process and detects if there is any of attack that is happing while execution .

2.9.3 Static and Runtime Analysis:

In this type of analysis, different researches had Chosen to combine the two aforementioned techniques to create a more effective and reliable solution to obtain a higher quality with a faster development and testing processes.

2.10 Suggested Hybrid Technique:

This section focuses on the main idea of the suggested hybrid technique for detecting and preventing SQLIAs.

2.10.1 Normal Data Exchanging Strategy:

There is much architecture to manage and to organize any data-driven systems, but the most common architecture that has been used is the three-tier architecture that depends on dividing the system into three tiers as follows:

1. Presentation Tier (a Web browser or rendering engine).

2. Logic Tier (a server code, such as C#, ASP, .NET, PHP, JSP, etc. …).

3. Storage Tier (a database such as Microsoft SQL Server, MySQL, Oracle, etc.).

4. Suggested Approach Strategy

The suggested approach is a runtime detection and prevention methodology that follows the same steps as the normal approach to exchange the queries between the architecture parties (Presentation-Logic-Storage), however, it provides an extra defense line on the Data-Tier to ensure that this side will not execute any abnormal codes that incase affect the system partially or Completely or it affects the hosted operating system and devices.

This approach is based on providing security controlling methodology on the database server side to ensures that all requested SQL queries from an inside or an outside the system are executed securely without any database fabrication or hacking. The suggested approach is based on different stages to reject any malicious query from being passed through the database engine before its execution process.

2.10.2 Replicate system databases:

For each database to be secured from SQLIAs, there should be a new replication database and it should contain a small amount of sample data.

Creating “database_Behaviors” database: The suggested approach should have a separate Database called “database_Behaviors” that contains all system database queries and their expected behaviors that have resulted from SQL queries execution in normal cases. This database is placed in the replicated instances.

2.10.3 Redirect SQL queries:

Any SQL query assigned to be executed in the target database will be initially delayed and replicated by the database engine then this replicated query is sent to the virtual database (Schema Replicated database). Hence, the original SQL query will be not executed yet in this stage and it will be delayed to a later stage.

2.10.4 Simple SQL syntax checking:

All SQL queries that are passing through the replicated database should also pass through multiple check processes before they move to the next step namely, "The execution process". The following list presents the checks processes that the SQL queries should pass through:

1. Encoding analysis: Before continuing to any next step the received SQL queries should be analyzed to determine the character encoding that has been used to write these queries. There are many techniques that can be used to do this analysis process such as "Automatic Identification of Language and Encoding"

2. Simple White-Box validation: The query should go through simple syntax validation and filtering for specific SQL reserved words especially those that use (EXECUTE, SHELL commands).

3. Parameters replacement: Any parameter that has been found in the SQL query should be replaced by an indexing parameter names. Such as (@par_1, @par_2 … @par _n).

2.10.5 Virtual execution:

After the SQL syntax checking process, the SQL query will be executed on the replicated database "Virtual Database" in which it is a process that is running simultaneously with the execution process, it monitors and traces the behaviors of the SQL query.

2.10.6 SQLIA Detection:

This stage is the most important stage in the suggested technique, its purpose is to detect whether the received SQL query is valid and expected query or not. The idea here is to catch the objects that have been affected by the current SQL query whatever the type of such objects and create a list of these objects to use them in the next step of this stage. The resulted list of affected objects will be compared with the “database Behaviors”.

If there is a query that handles all of the listed objects with the same type of behavior that is detected from the previous step then this behavior query will be added to a new list (Expected Queries). Any resulted behavior that is detected as a suspicious should be rejected and deleted from the actual database instance execution queue, otherwise the query will be transferred to the actual database instance for being executed [22]

2.11 Related Work

A number of electronic journals articles from IEEE journals and from ACM, and gathered some information from web sites to gain sufficient knowledge about SQL injection attacks. Following are the papers from which I covered different important strategies to prevent and detect SQL injection attacks.

The Paper [23]

The paper looks into frequent SQL injection attack forms, their mechanisms, and a way of identifying them based on the SQL query's existence. They propose a comprehensive framework for determining the effectiveness of techniques that address certain issues following the essence of the attack, using hybrid (Statistic and dynamic) and machine learning. An extensive examination of the model based on a test set indicates that the Hybrid and ANN approaches outperform Naive Bayes, SVM, and Decision trees in terms of accuracy in classifying injected Queries. However, when it came to web loading time during testing, Nave Bayes outperformed. The Hybrid Method improved the accuracy of SQL injection attack prevention, according to the test findings. Although They used a limited dataset for training and testing in our study, it is advised that the dataset be expanded and the model be tested in a real-world setting.

The Paper [24]

The paper, They propose a SQL injection detection method that does not rely on background rule base by using a natural language processing model and deep learning framework on the basis of comprehensive domestic and international research. The method can improve the accuracy and reduce the false alarm rate while allowing the machine to automatically learn the language model features of SQL injection attacks, greatly reducing human intervention and providing some defense against 0day attacks that never occur. The paper implements a SQL injection detection system based on a deep learning framework and combining data preprocessing and lexical analysis techniques. The experiments show that the system can detect first-order SQL injection attacks more accurately and efficiently

The Paper [25]

This paper successfully proposed a new comprehensive method using an improved parameterized stored procedure to overcome the three issues highlighted in preventing web application from SQL injection attack ; In this study, a stored procedure is constructed that consists of a comprehensive method to address these three issues which are built in the code. To prove the effectiveness, the proposed method was evaluated through three approach of attack simulation, namely using SQLMap software, Netsparker and web browser. The experiments conclude that SQL injection does not successfully penetrate web applications and databases when the proposed method is implemented hence able to overcome the limitations faced by the existing methods. That indicates that the SQL injection prevention method developed has successfully prevents SQL injection from occurring. The proposed method is also evaluated from the perspective of time used and its impact on the overall system. Evaluations using Microsoft SQL Server Client Statistics and SQLQueryStress software have concluded that although there are slight differences in time processing, the proposed method uses a shorter query processing time when compared with dynamic SQL. It can be concluded that the SQL injection prevention method used does not generate high overhead that may lead to high response time.

.

The Paper [26]

A novel approach to detect and prevent SQL injection and XSS attacks is presented in this paper. The various types and patterns of the attacks were first studied, then a parse tree was designed to represent the patterns. Based on the identified patterns, a filter() function was formulated using the KMP string matching algorithm. The formulated filter() function detects and prevents any form of SQL injection and XSS attack; The test results show that the technique can successfully detect and prevent the attacks, log the attack entry in the database, block the system using its Mac Address to prevent further attack, and issue a blocked message

The Paper [27] presented some different papers with several proposed methods and tools to detect and prevent the SQL injection attack to be able to get a wide range of ideas in this way and compare them against each other. They defined a methodology in our research to be able to get more information from each paper. So, after that it was much easier to compare the methods and analyzing them.

They focused on main parts and phases of each method proposed in each paper and then They found the advantages and limitations of them at the end. The techniques that is used in this paper :

Analysis and Monitoring for Neutralizing(AMNESIA) , Preventing SQL Injection Attacks (SQLrand), Syntactic and Semantic Analysis for Automated Testing against SQL Injection(Sania), A specification–based Approach for SQL-injection Detection(SQL-IDS) , Location-specific signatures prevent SQL injection attacks(SDriver) , A heuristic-based approach for detecting SQL injection vulnerabilities in Web applications, Learning Fingerprints for a Database Intrusion Detection System.

2.1 Summary of related works

Name of Paper	Date	Author	Techniques	result
Attacks on SQL Injection and Developing Compressive Framework Using a Hybrid and Machine Learning Approach **[23]**	February 8th, 2022	Fitsum Gizachew Deriba , Tsegay Mullu Kassa , Wubetu Baud Demi lie	using hybrid and machine learning techniques	They employed three injection parameters in various forms in this study. The first is through a user input field, which allows a web application used to request information from a backend database using HTTP POST and GET, and the second is through cookies, which may be used to restore a client's state information when they return to a Web application. If a Web application uses the contents of cookies to construct SQL queries, an attacker can exploit this vulnerability to change cookies and submit them to the database server. Finally, by analyzing session usage information and Recognizing browsing behaviors, a server variable can be created. Because attackers can forge the values that are placed in HTTP and network headers by entering malicious input into the client-end of the application or by crafting their request to the server, if these variables are logged to a database without sanitization, that could result in SQL injection vulnerability. That attack parameter includes all of the attack types mentioned in the paper .

SQL Injection Attack Detection and Prevention Techniques Using Deep Learning[24]	2021	Ding Chen*, Qiseng Yan, Chunwang Wu, Jun Zhao	based on a deep learning framework and combining data preprocessing and lexical analysis techniques	The paper implements a SQL injection detection system based on a deep learning framework and combining data preprocessing and lexical analysis techniques. The experiments show that the system can detect first-order SQL injection attacks more accurately and efficiently. Subsequent research will focus on advanced SQL injection attack methods, such as second-order injection and hybrid attacks.
A Method to Prevent SQL Injection Attack using an Improved Parameterized Stored Procedure [25]	2021	Kamsuriah Ahmad , Mayshara Karim	parameterized stored procedure	In The paper , a stored procedure is constructed that consists of a comprehensive method to address these three issues which are built in the code. To prove the effectiveness, the proposed method was evaluated through three approach of attack simulation, namely using SQL Map software, Nets parker and web browser. The experiments conclude that SQL injection does not successfully penetrate web applications and databases when the proposed method is implemented hence able to overcome the limitations faced by the existing methods. This indicates that the SQL injection prevention method developed has successfully prevents SQL injection from occurring. The proposed method is also evaluated

				from the perspective of time used and its impact on the overall system. Evaluations using Microsoft SQL Server Client Statistics and SQL QueryStress software have concluded that although there are slight differences in time processing, the proposed method uses a shorter query processing time when compared with dynamic SQL. It can be concluded that the SQL injection prevention method used does not generate high overhead that may lead to high response time. This study can be further improved by focusing the prevention of SQL injection in network systems. There are various factors to be considered such as the types of server used, network traffic and the attacks from various sources. More efforts are needed and further enhancements are required to improve database security
A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm [26]	2020	Oluwakemi Christiana Abikoye, Abdullahi Abubakar, Ahmed Haruna Dokoro , Oluwatobi Noah Akande and Aderonke Anthonia Kayode	Knuth-Morris-Pratt (KMP) string matching algorithm	The test results show that the technique can successfully detect and prevent the attacks, log the attack entry in the database, block the system using its Mac Address to prevent further attack, and issue a blocked message. A comparison of the proposed technique with existing techniques revealed that

				the proposed technique is more efficient because it is not limited to a particular form of attack, and it can handle different forms of SQL injection and XSS attacks.
SQL Injection Attacks: Detection And Prevention Techniques [27]	2019	Rania Alsahafi	SQLrand , SQL-ID , Sania , SDriver	Many different approaches have been proposed to detect and in consequence prevent SQL injection attacks. Some of them are trying to find vulnerabilities in the web applications and then trying to solve the problems; some others are focusing on the user inputs and verifying them based on some predefined patterns and algorithms. Some of the methods and tools are relying on some others and some of them have brand new ideas. We tried to study as many different methods as possible.

Table (2.1) : summary of related work

Chapter Three

Methodology

3.1 Introduction:

In our example hack showed you how to bypass the login page: a huge security flaw for a banking site. More complex attacks will allow an attacker to run arbitrary statements on the database. In the past, hackers have used injection attacks to: Extract sensitive information, like Social Security numbers, or credit card details. Enumerate the authentication details of users registered on a website, so these logins can be used in attacks on other sites. Delete data or drop tables, corrupting the database, and making the website unusable. Inject further malicious code to be executed when users visit the site. SQL injection attacks are astonishingly common. Major companies like Yahoo and Sony have had their applications compromised. In other cases, hacker groups targeted specific or wrote scripts intended to harvest authentication details. Not even security firms are immune [28]

3.2 The Frame Work :

1. Study the risk in fine details.
2. Determine the expect risk.
3. Understood the vulnerability nature.
4. Determine the impact of that risk.
5. Suggest policies and applying it.
6. Measure and write the result matrix.

3.3 Proposed Method Diagram:

The proposed model based on dynamic analyzer that works as:

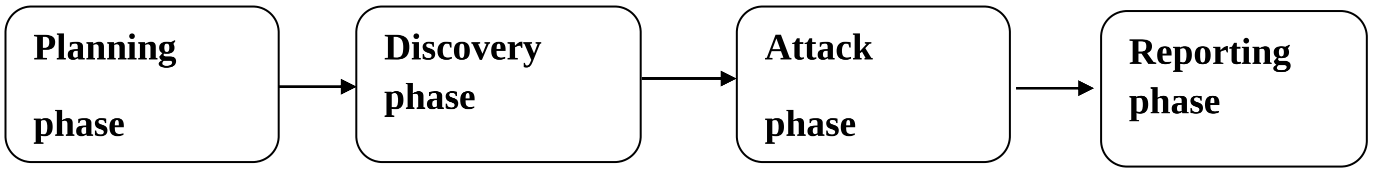


Figure (3.1) : Framework process

3.3.1 Planning Phase:

This is the First Phase in Methodology, divided into two Steps:

1. Scope and strategy of the assignment is determined.

2. Existing security policies, standards are used for defining the scope.

3.3.2 Discovery Phase:

The Second Phase divided into Four Steps:

1. Collect as much information as possible about the system including data in the system, user names and even passwords. This is also called as FINGERPRINTING.
2. Design the system by using Asp.net language & Microsoft SQL Server Management Studio database.
3. Scan and Probe into the port.
4. Check for vulnerabilities of system by using Two method for prevention and detection SQLIA called: Validating user input, Using Prepared Statements (with Parameterized Queries) SQLMAP,WPSCAN,COMMIX

3.3.3 Attack phase:

At this very important Stage of my research Methodology :

1. Find exploits for various vulnerabilities that is need necessary security Privileges to exploit the system, also using LINUX KHALI operating system platform and there tools that is used in security such as: **SQLMAP,WPSCAN,COMMIX**

3.3.4 Reporting phase:

This is the last Stage of the research and Result:

1. report must contain detailed findings
2. Risks of vulnerabilities found and their Impact on business
3. Recommendation and conclusion, if any

3.3 Flow Chart:

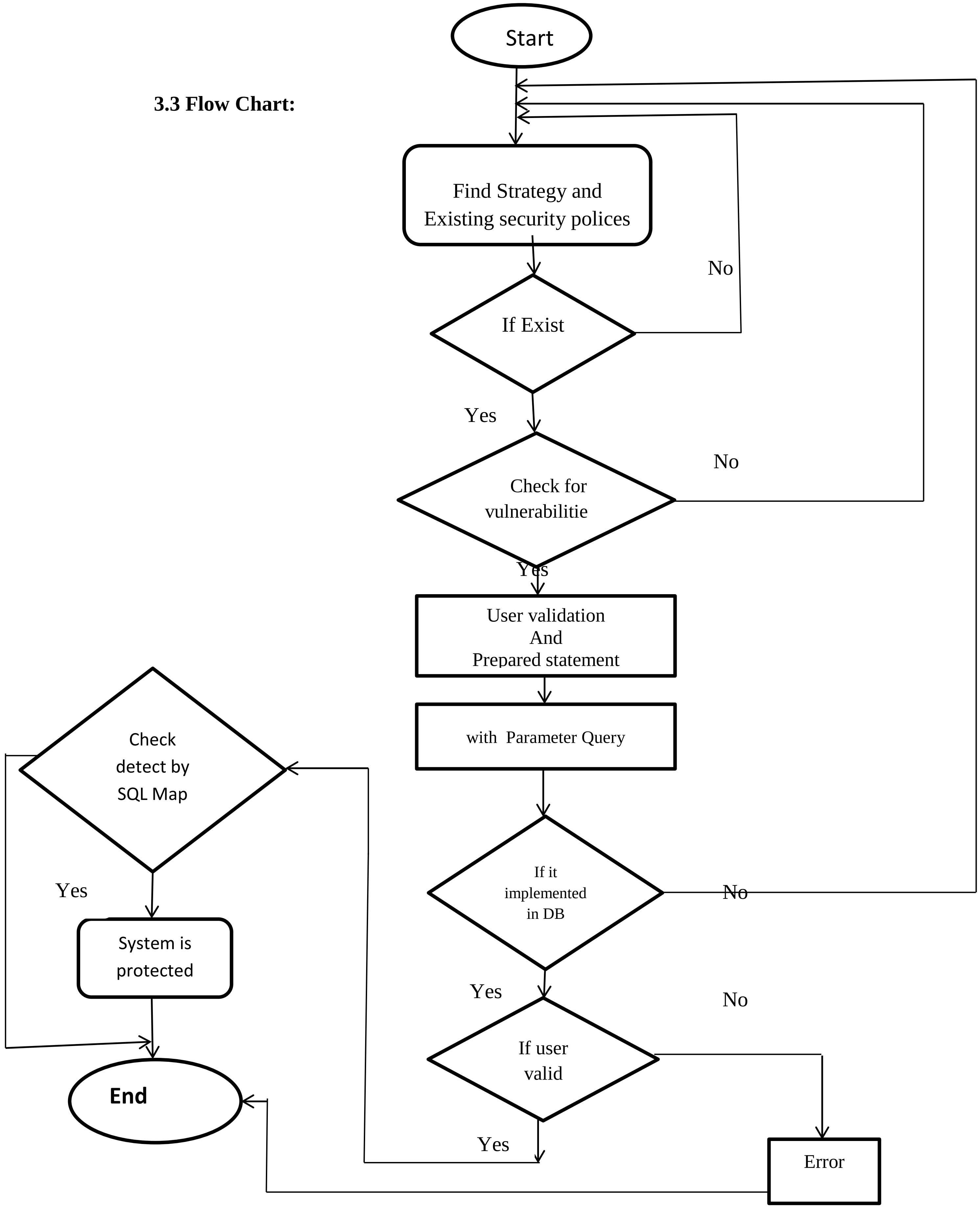


Figure (3.2): flow chart of methodology

3.5 Device Requirements:

1. Device types
2. Performance characteristics
3. Internet

3.5.1 Device types:

1. Generic computer devices : PCs and Laptops .
2. Servers: storage servers and application servers.

3.5.2 Performance characteristics:

1. Storage
2. Processor
3. Memory
4. Operating System
5. Device driver

3.6 Management Tasks :

1. Even processing
2. Monitoring
3. Attack detection and troubleshooting
4. Accounting of vulnerability
5. Security polices and information of the system

Chapter Four

Analysis and Implementation

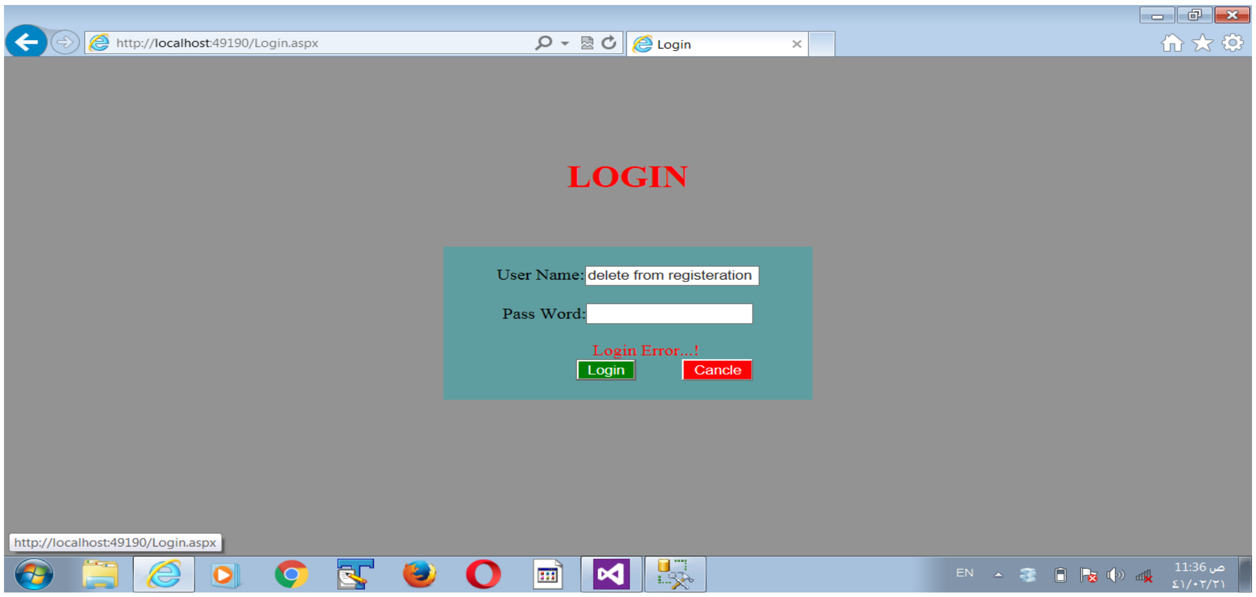
4.1 Introduction :

This Chapter present the Design of webpages by ASP.NET and connected to data base (SQL Server management studio) , I build two web sites the first one isn't applying the prevention policy, but the second applying policy and third one acts as ready package to compare between them. Also applies two of these methods and evaluated by normal methods. The implements done through web site include three page linked by hyper link showed in figures below:

4.2 The first page:

acts as **LOGIN** page which is the attacker target. The page consist two text box as input of username, his password and bottom. This page build firstly as pure page without apply any prevention method. The same page repeated as improved page by applying the more than one prevention methods .Each field must fill by the same data types as defined.

The operation done through this page illustrated in The Figure (4.1):


http://localhost:49190/Login.aspx
Login
LOGIN
User Name:
Pass Word:
Login
Cancle
http://localhost:49190/Login.aspx

delete from registeration

Login Error...!

Figure (4.1): The page login1

The figure (4.2) : Shows The same page repeated as improved page by applying the more than one prevention methods.

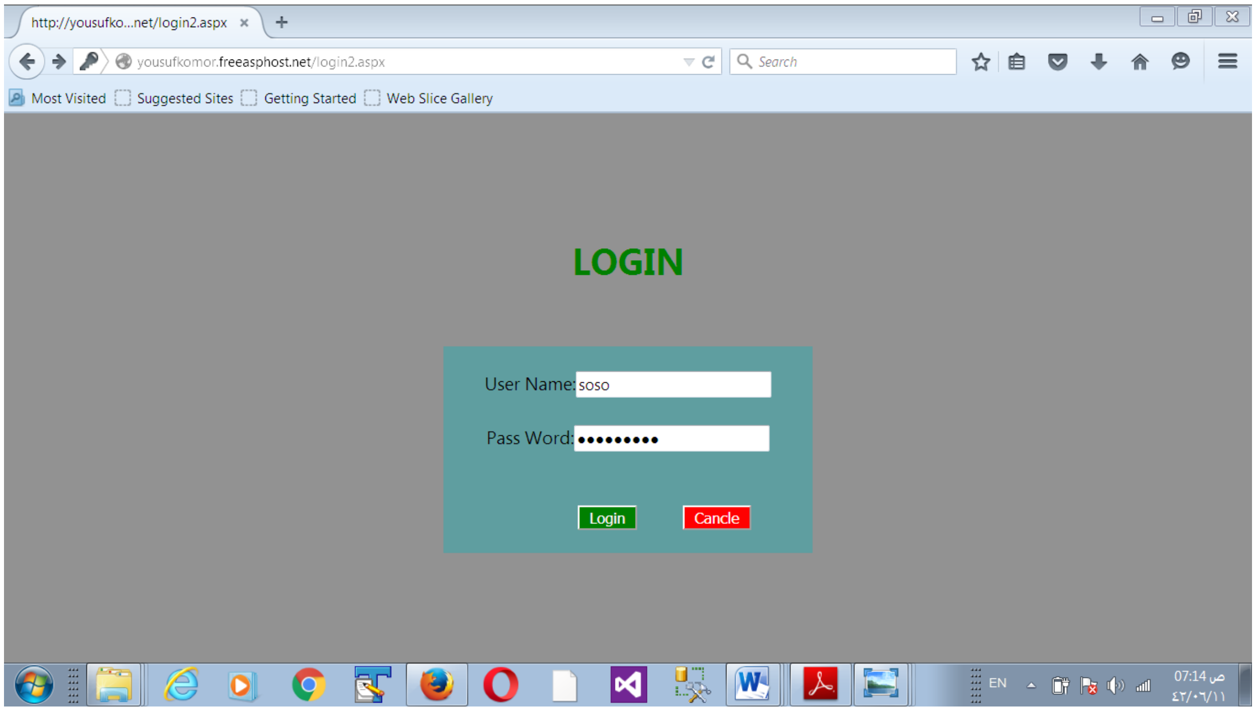


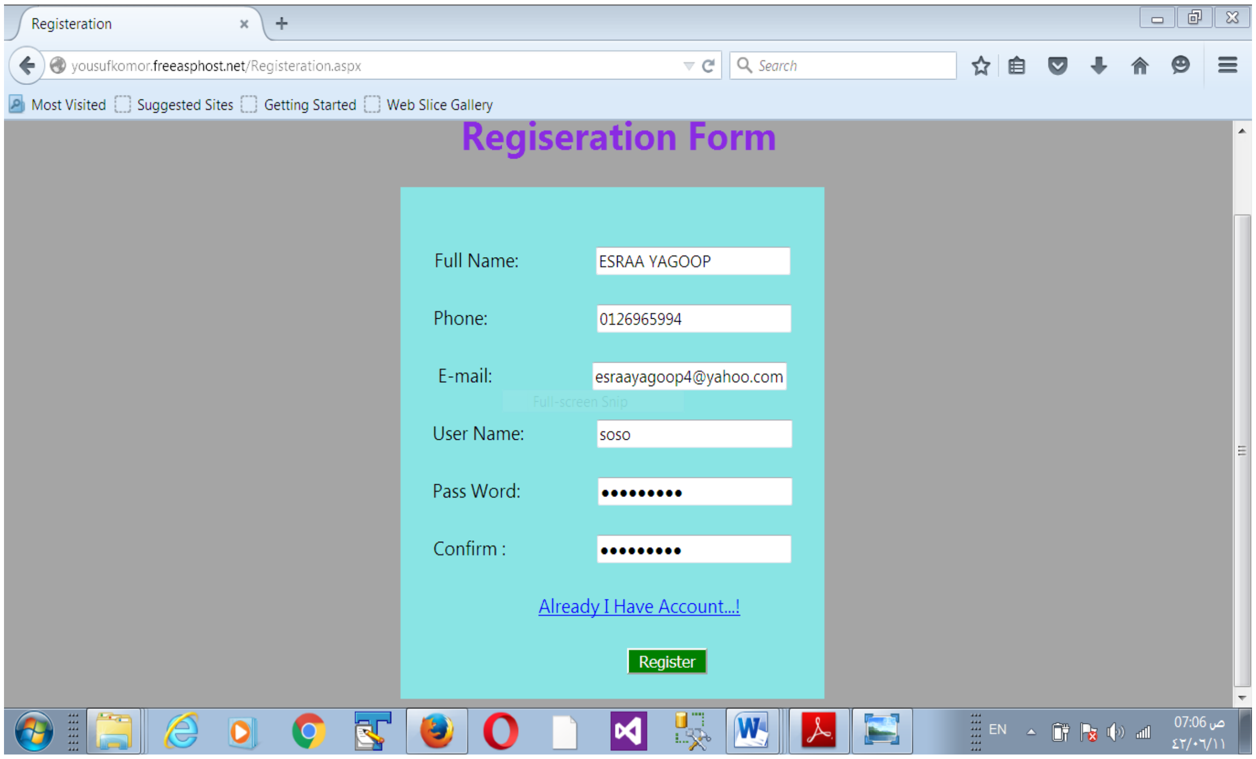
Figure (4.2): The page Login2

4.3 The second page:

The second page is **REGISTRATION** page which is include the (first name, last name, address, e-mail address, phone number, password and confirm password).

The data base build on SQL Server include one table which is registration table.

The Figure (4.3) : Shows registration form ,the user write all this information correctly Then click the button register


Registeration

yousufkomor.freeasphost.net/Registeration.aspx

Most Visited | Suggested Sites | Getting Started | Web Slice Gallery

Regiseration Form

Full Name:	ESRAA YAGOOP
Phone:	0126965994
E-mail:	esraayagoop4@yahoo.com
User Name:	soso
Pass Word:	•••••••••
Confirm :	•••••••••

Already I Have Account...!

Register

Figure (4.3): Registration Form

4.4 The last page:

The last page is CONFIRM page which is include welcome express to insure that right operation executed without any error and attack operation may be execute where in the case no apply any policy of SQL injection prevention. Shows Figure (4.4)

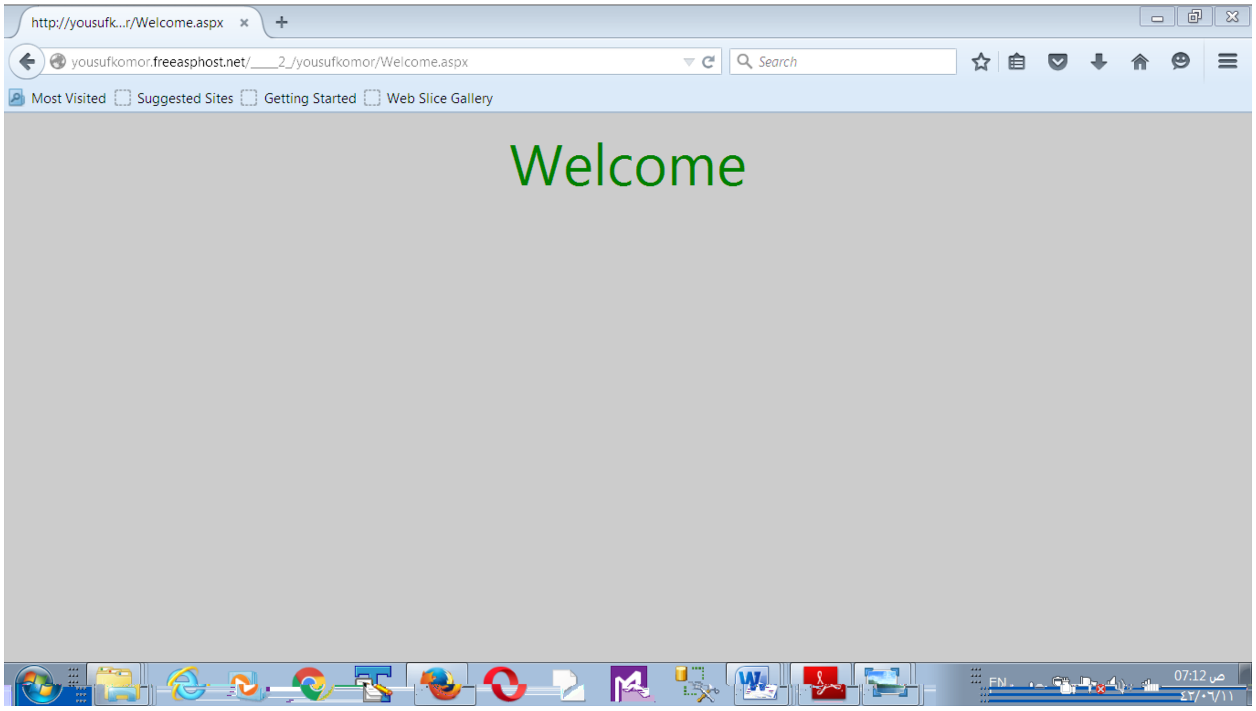
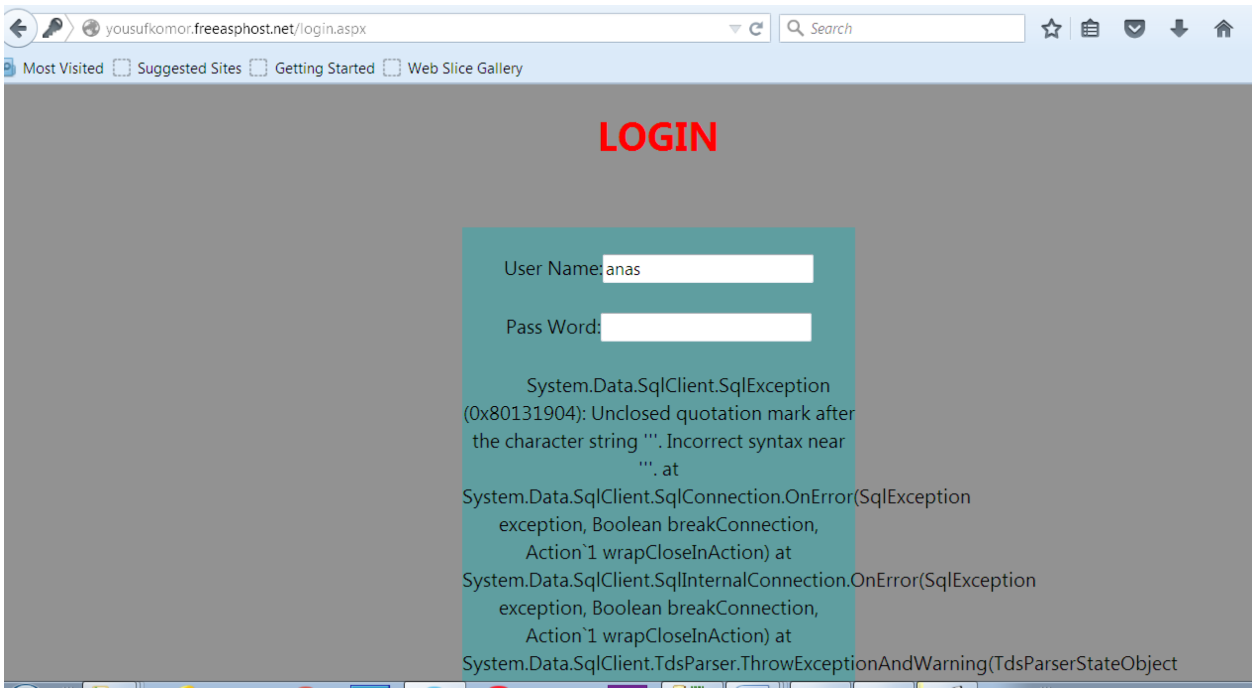

http://yousufk...r/Welcome.aspx
yousufkomor.freeasphost.net/____2_/yousufkomor/Welcome.aspx
Search
Most Visited
Suggested Sites
Getting Started
Web Slice Gallery
Welcome

Figure (4.4): Welcome Page

4.5 The login page (not prevention)

the figure(4.5): shows the login page that is not prevention for SQLIA, when the user write the username and password in correctly ,the page not response and print this message but it define what is happen in the system and shows the vulnerability in system



LOGIN

User Name: anas

Pass Word:

```
System.Data.SqlClient.SqlException
(0x80131904): Unclosed quotation mark after
the character string '''. Incorrect syntax near
'''. at
System.Data.SqlClient.SqlConnection.OnError(SqlException
exception, Boolean breakConnection,
Action`1 wrapCloseInAction) at
System.Data.SqlClient.SqlInternalConnection.OnError(SqlException
exception, Boolean breakConnection,
Action`1 wrapCloseInAction) at
System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject
```

Figure (4.5): login1 not prevention

4.6 Applied the Method: User Validation and Prepared Statements to prevention SQLIA:

The figure(4.6): shows the message Login Error , here in this page applied the code for prevention SQLIA by using the method s user validation and prepared statement

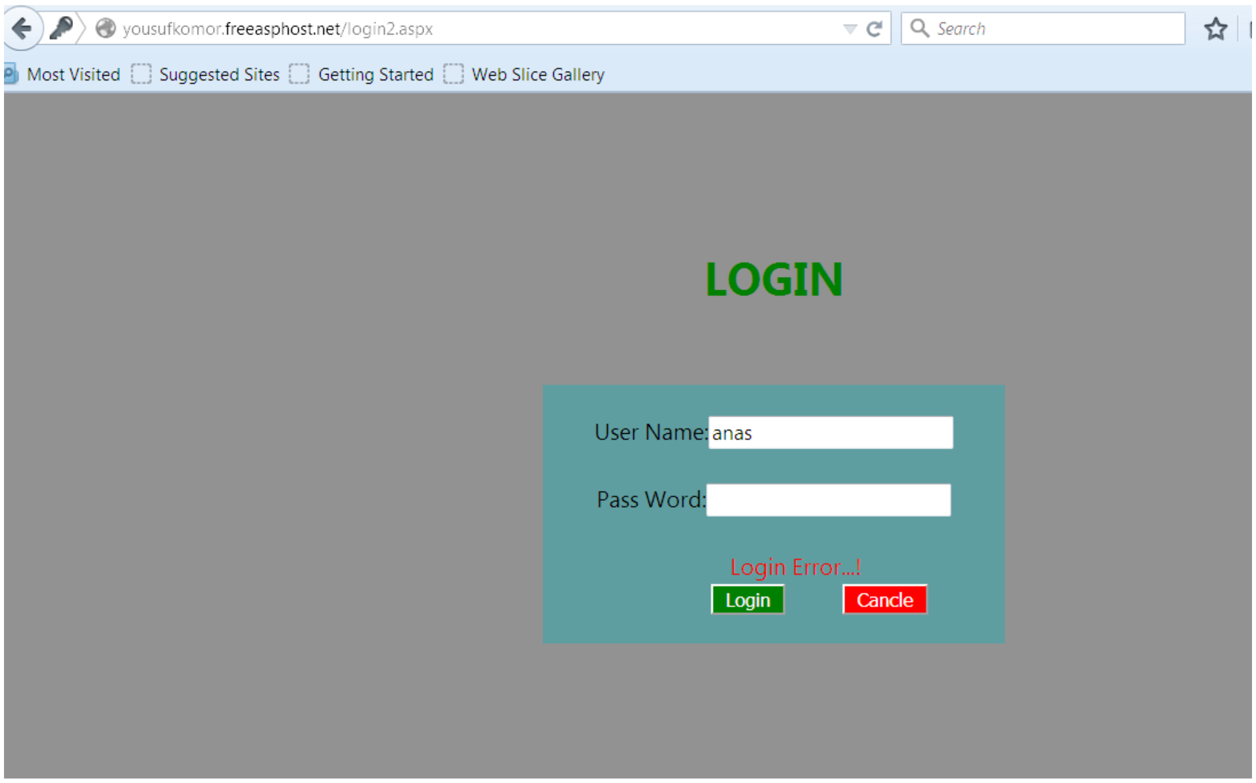


Figure (4.6):login2 prevention

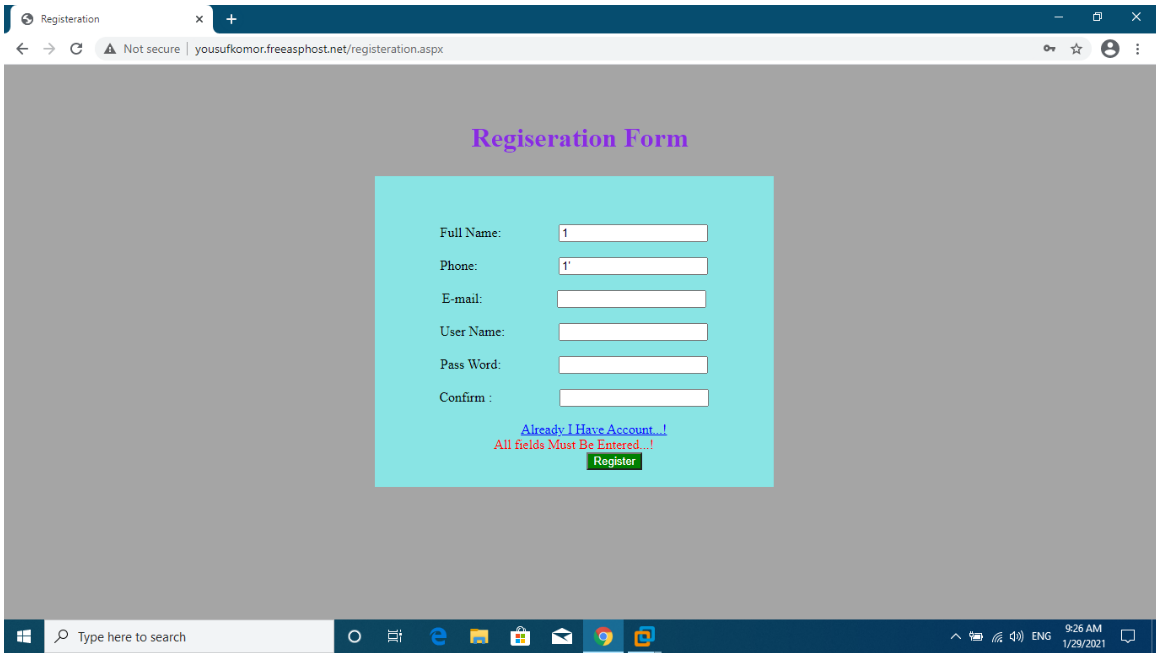
4.7 The implemented Code by using Prepared Statements Method (with Parameterized Queries) :

```
SqlConnection(connectionString);
                    //string myquery = "select * from
registerartion  where username ='" + txtUserName.Text + "' and
password  ='" + txtPassword.Text + "'";

string myquery = "select * from registerartion  where username
=@username    and   password =@password";

con.Open();

SqlCommand cmd = new SqlCommand(myquery, con);

cmd.Parameters.AddWithValue("@username",txtUserName1.Text);
cmd.Parameters.AddWithValue("@password", txtPassword1.Text);
```

Figure (4.7): The implemented Code

4.8 Using SQL chars For Detection SQLIA:

The figure(4.8) ,(4.9),(4.10),(4.11),(4.12) Using SQL chars like(\ " ‘) that it used for detection SQLIA in Web Pages ,firstly filling the form registration by added (‘)at ended of number, and then complete the registration; also Shows the welcome page that’s indicates the success of the operation.



Regiseration Form

Full Name: 1

Phone: 1'

E-mail:

User Name:

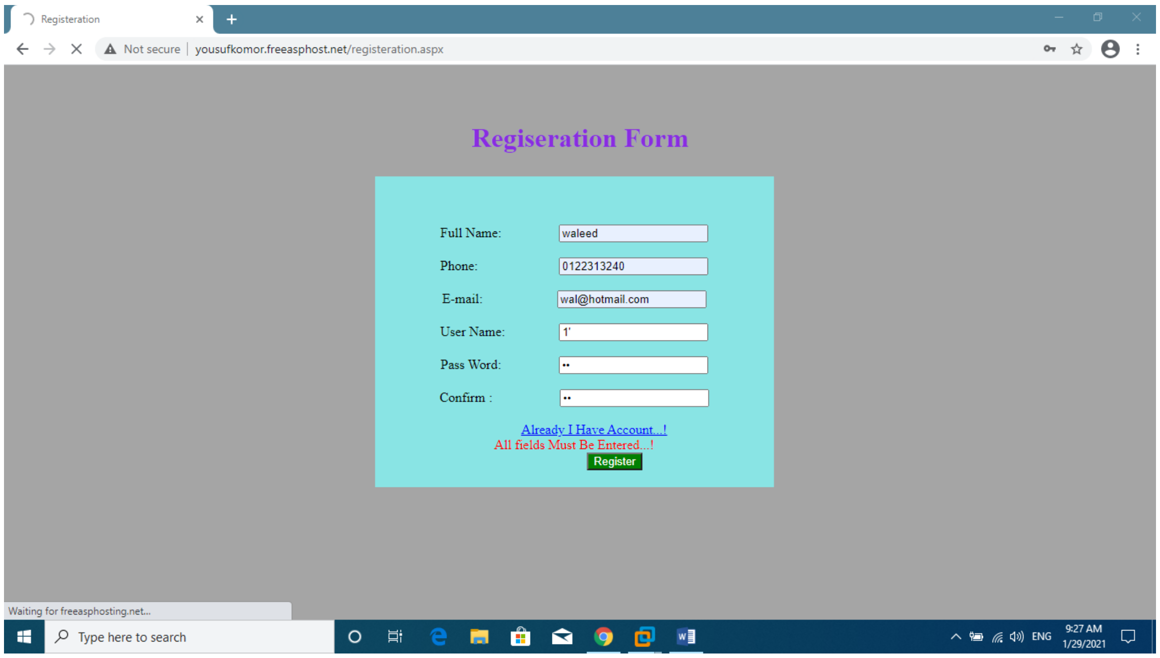
Pass Word:

Confirm :

Already I Have Account...!

All fields Must Be Entered...!

Figure (4.8) : using special char



Regiseration Form

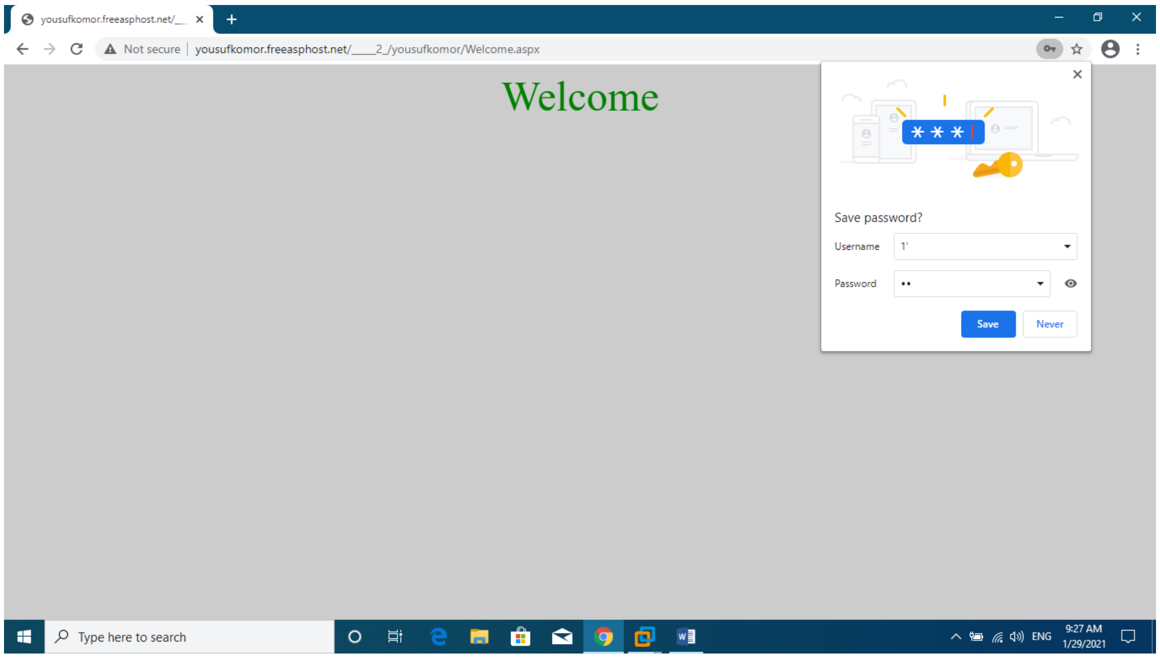
Full Name:	waleed
Phone:	0122313240
E-mail:	wal@hotmail.com
User Name:	1'
Pass Word:	••
Confirm :	••

Already I Have Account...!

All fields Must Be Entered...!

Registeration
Not secure
yousufkomor.freeasphost.net/registeration.aspx
Register
Waiting for freeasphosting.net...
Type here to search
9:27 AM
1/29/2021

The Figure (4.9): complete form



Welcome

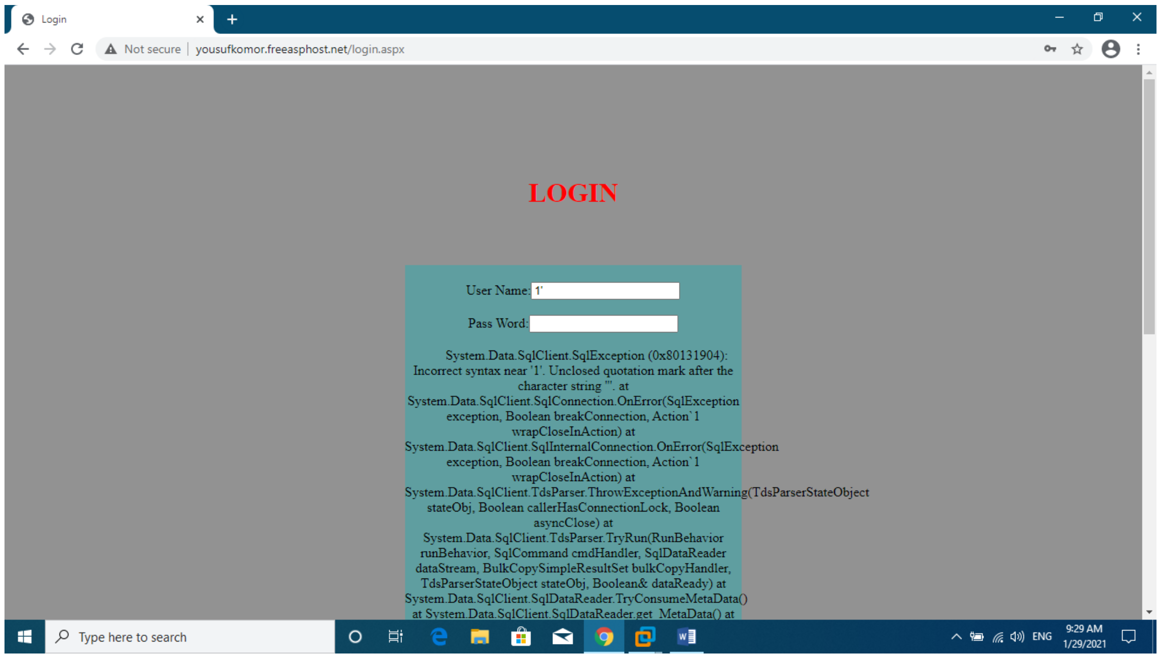
Save password?

Username	1'
Password	••

yousufkomor.freeasphost.net/___
Not secure
yousufkomor.freeasphost.net/____2_/yousufkomor/Welcome.aspx
Save
Never
Type here to search
9:27 AM
1/29/2021

Figure (4.10): The welcome page

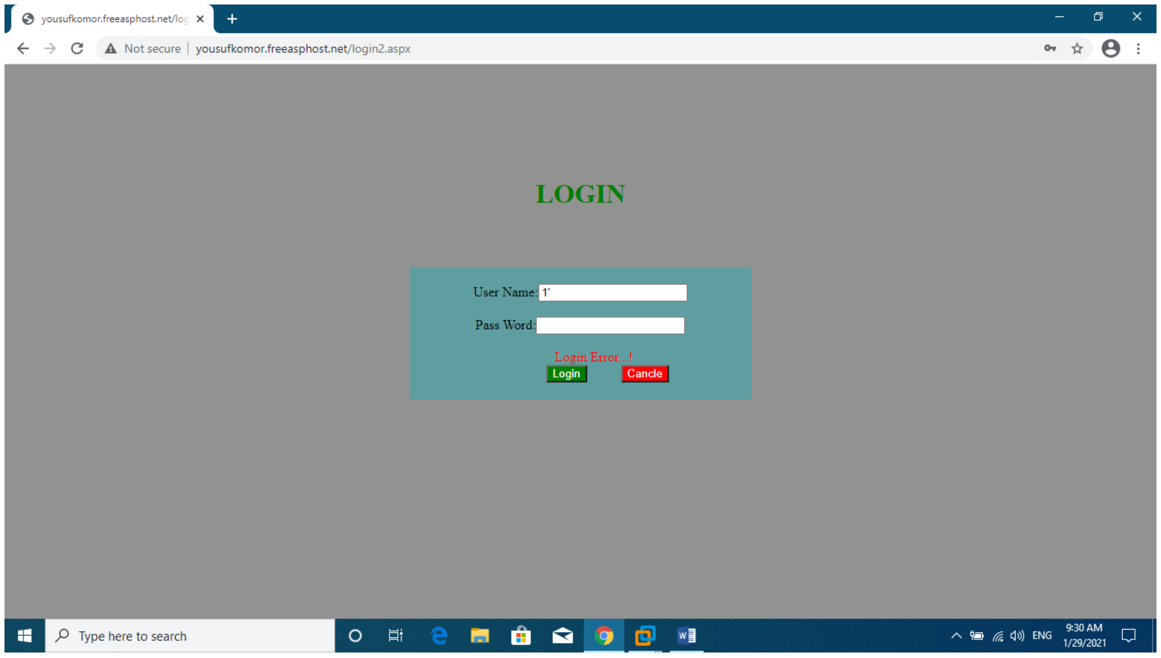
The figure (4.11): Shows the long message that's indicates here founded and Detected SQLIA



```
System.Data.SqlClient.SqlException (0x80131904):
Incorrect syntax near '1'. Unclosed quotation mark after the
character string ''. at
System.Data.SqlClient.SqlConnection.OnError(SqlException
exception, Boolean breakConnection, Action`1
wrapCloseInAction) at
System.Data.SqlClient.SqlInternalConnection.OnError(SqlException
exception, Boolean breakConnection, Action`1
wrapCloseInAction) at
System.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject
stateObj, Boolean callerHasConnectionLock, Boolean
asyncClose) at
System.Data.SqlClient.TdsParser.TryRun(RunBehavior
runBehavior, SqlCommand cmdHandler, SqlDataReader
dataStream, BulkCopySimpleResultSet bulkCopyHandler,
TdsParserStateObject stateObj, Boolean& dataReady) at
System.Data.SqlClient.SqlDataReader.TryConsumeMetaData()
at System.Data.SqlClient.SqlDataReader.get_MetaData() at
```

Figure (4.11):login1 by using SQL char

The figure (4.12): Shows the Error message (Login Error) that's indicates this web page prevention of SQLIA and can't permission to login it .



figure(4.12): login2 by using SQL char

4.9 Applied the Script for detect SQLIA:

Using the script below for detected SQLIA:

<script> document .location= http://yousufkomor.freeasphost.net/login.aspx </script>

<script> alter (1) </script>

The SQLIA founded in Header by using (Get, Post function), textbox and payload.

figure(4.13),(4.14) : Shows the execution of the script and the Result message that's said

A potentially dangerous Request : form value was detected from the client…

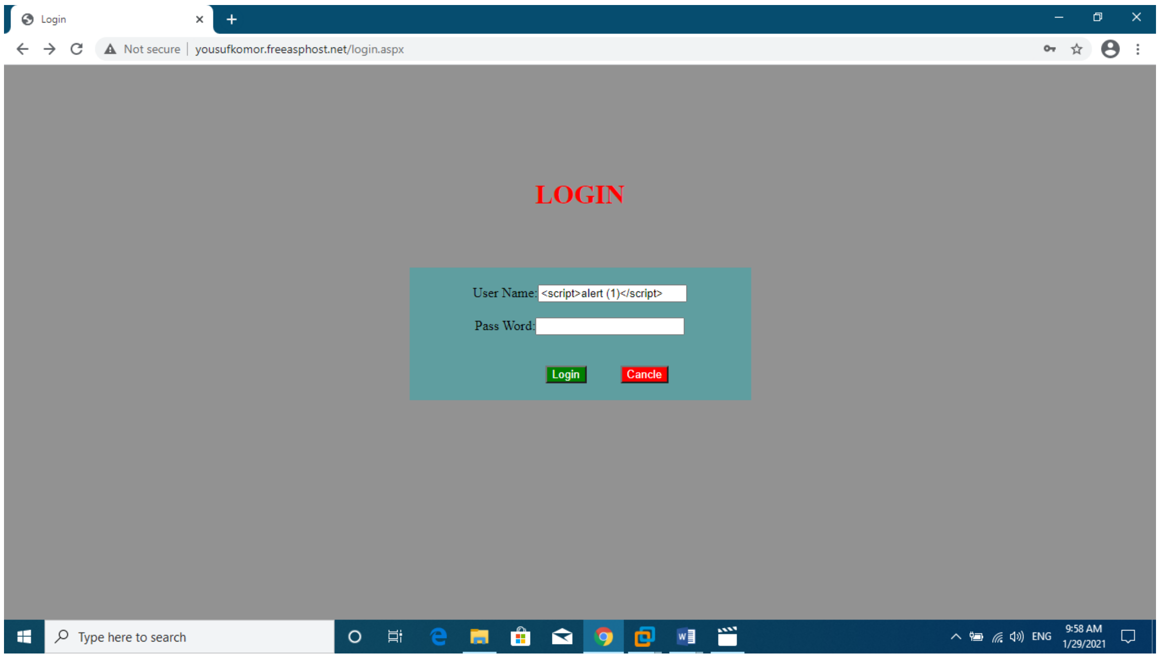
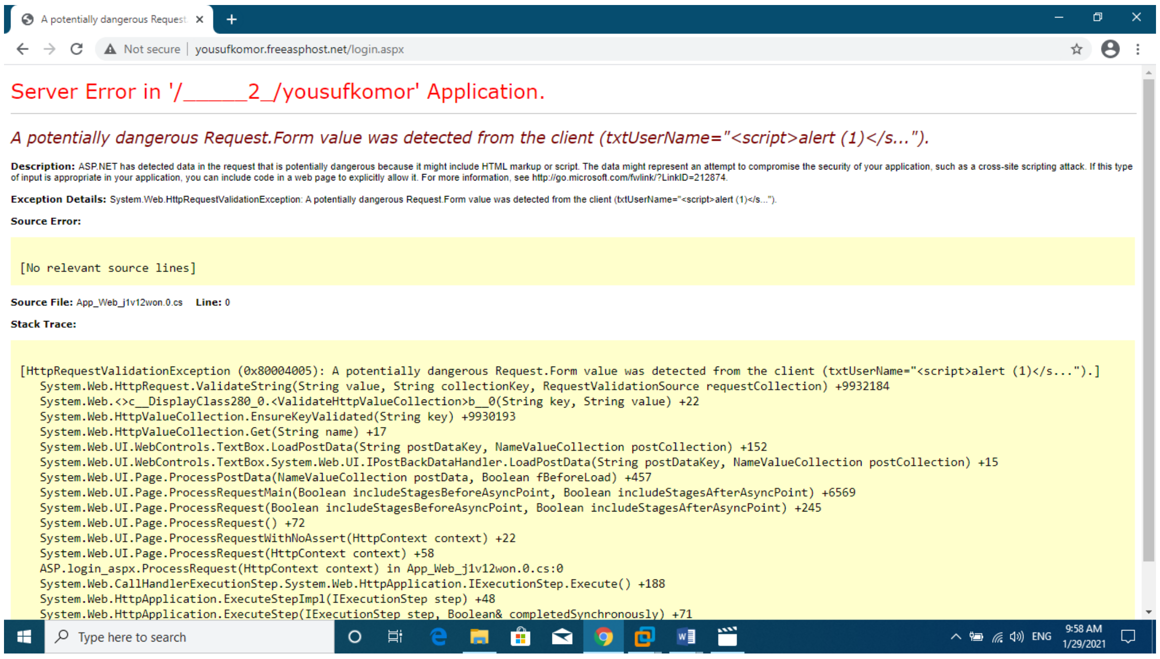

Login
Not secure | yousufkomor.freeasphost.net/login.aspx
LOGIN
User Name: <script>alert (1)</script>
Pass Word:
Login
Cancle
Type here to search
9:58 AM
1/29/2021

Figure (4.13): script alter at login 1



Server Error in '/______2_/yousufkomor' Application.

A potentially dangerous Request.Form value was detected from the client (txtUserName="<script>alert (1)</s...").

Description: ASP.NET has detected data in the request that is potentially dangerous because it might include HTML markup or script. The data might represent an attempt to compromise the security of your application, such as a cross-site scripting attack. If this type of input is appropriate in your application, you can include code in a web page to explicitly allow it. For more information, see http://go.microsoft.com/fwlink/?LinkID=212874.

Exception Details: System.Web.HttpRequestValidationException: A potentially dangerous Request.Form value was detected from the client (txtUserName="<script>alert (1)</s...").

Source Error:

```
[No relevant source lines]
```

Source File: App_Web_j1v12won.0.cs **Line:** 0

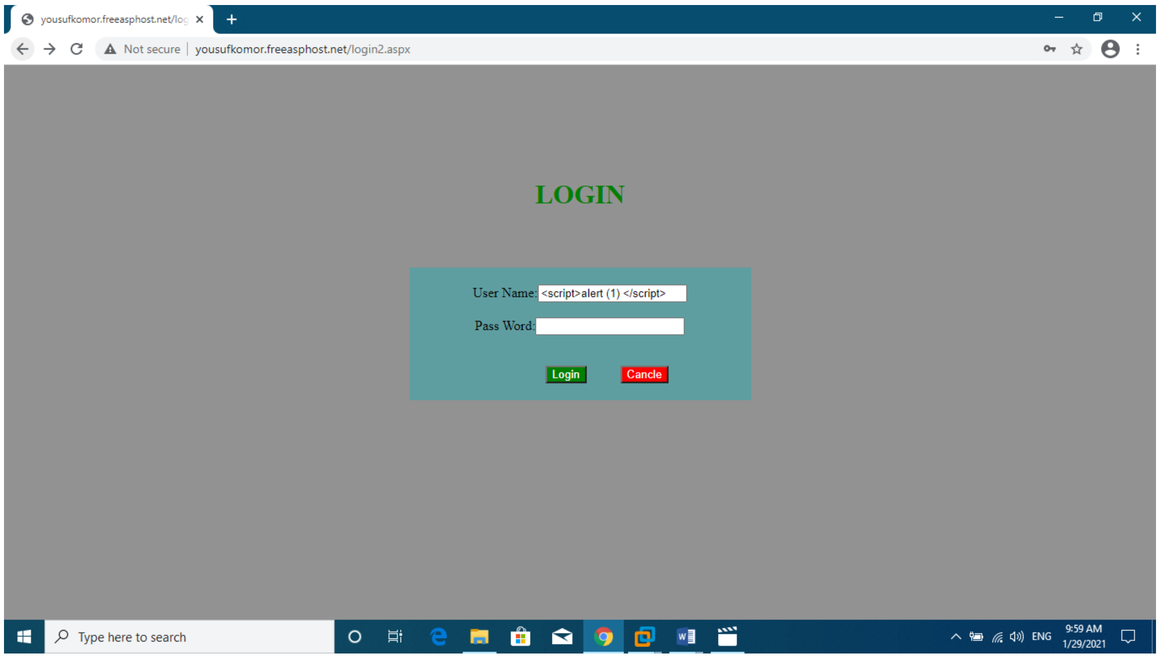
Stack Trace:

```
[HttpRequestValidationException (0x80004005): A potentially dangerous Request.Form value was detected from the client (txtUserName="<script>alert (1)</s...").]
   System.Web.HttpRequest.ValidateString(String value, String collectionKey, RequestValidationSource requestCollection) +9932184
   System.Web.<>c__DisplayClass280_0.<ValidateHttpValueCollection>b__0(String key, String value) +22
   System.Web.HttpValueCollection.EnsureKeyValidated(String key) +9930193
   System.Web.HttpValueCollection.Get(String name) +17
   System.Web.UI.WebControls.TextBox.LoadPostData(String postDataKey, NameValueCollection postCollection) +152
   System.Web.UI.WebControls.TextBox.System.Web.UI.IPostBackDataHandler.LoadPostData(String postDataKey, NameValueCollection postCollection) +15
   System.Web.UI.Page.ProcessPostData(NameValueCollection postData, Boolean fBeforeLoad) +457
   System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAfterAsyncPoint) +6569
   System.Web.UI.Page.ProcessRequest(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAfterAsyncPoint) +245
   System.Web.UI.Page.ProcessRequest() +72
   System.Web.UI.Page.ProcessRequestWithNoAssert(HttpContext context) +22
   System.Web.UI.Page.ProcessRequest(HttpContext context) +58
   ASP.login_aspx.ProcessRequest(HttpContext context) in App_Web_j1v12won.0.cs:0
   System.Web.CallHandlerExecutionStep.System.Web.HttpApplication.IExecutionStep.Execute() +188
   System.Web.HttpApplication.ExecuteStepImpl(IExecutionStep step) +48
   System.Web.HttpApplication.ExecuteStep(IExecutionStep step, Boolean& completedSynchronously) +71
```

A potentially dangerous Request
Not secure | yousufkomor.freeasphost.net/login.aspx
Type here to search
ENG
9:58 AM
1/29/2021

Figure (4.14): The result of the Script

The figure (4.15),(4.16) : Shows the Error message (Login Error) that's indicates this web page prevention of SQLIA and can't permission to login it and execute the same Script in login 1



LOGIN

User Name: <script>alert (1) </script>

Pass Word:

yousufkomor.freeasphost.net/log
Not secure | yousufkomor.freeasphost.net/login2.aspx
Login
Cancle
Type here to search
ENG
9:59 AM
1/29/2021

Figure (4.15): script alter at login 2

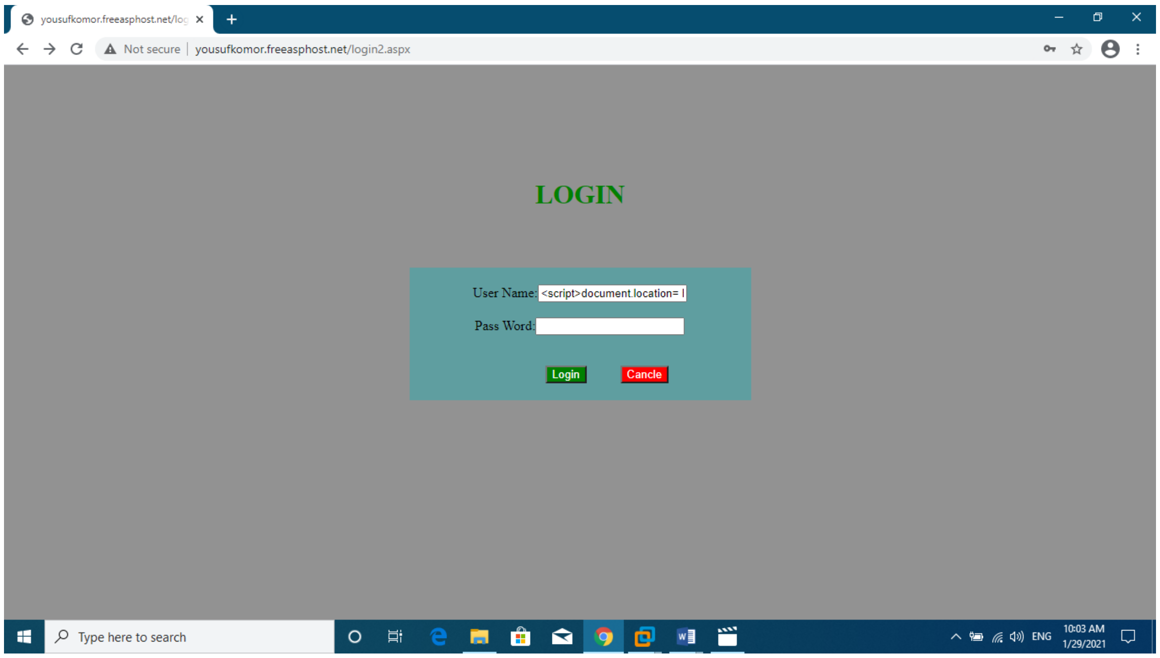

yousufkomor.freeasphost.net/log
Not secure
yousufkomor.freeasphost.net/login2.aspx
LOGIN
User Name:
<script>document.location= I
Pass Word:
Login
Cancle
Type here to search
ENG
10:03 AM
1/29/2021

Figure (4.16): script document. location   at login 2

4.2 Comparing Table:

Property **methods**	**Ability Detecting and Prevent.**	**Implementation.**	**Protect Efficiency.**
Updating your system.	May Prevent.	Easy.	Weak and dependency on the updating if not done.
Hiding info from the error message.	Prevent Only.	Needs some knowledge of errors messages.	Best for analysis area.
Limiting privileges.	Prevent Only.	Need recognized methods.	Weak, for other allowed.
Using Stored Procedures.	Detect and Prevent.	Relay on the prepared statements.	Weak, prevent done in data base.
Validating user input.	Detect and Prevent.	Require user input known.	More efficient acts as filter.
Using Prepared Statements.	Detect and Prevent.	Requires all states of the hacker that perform operations.	Full control and prevention if implies with user validating.

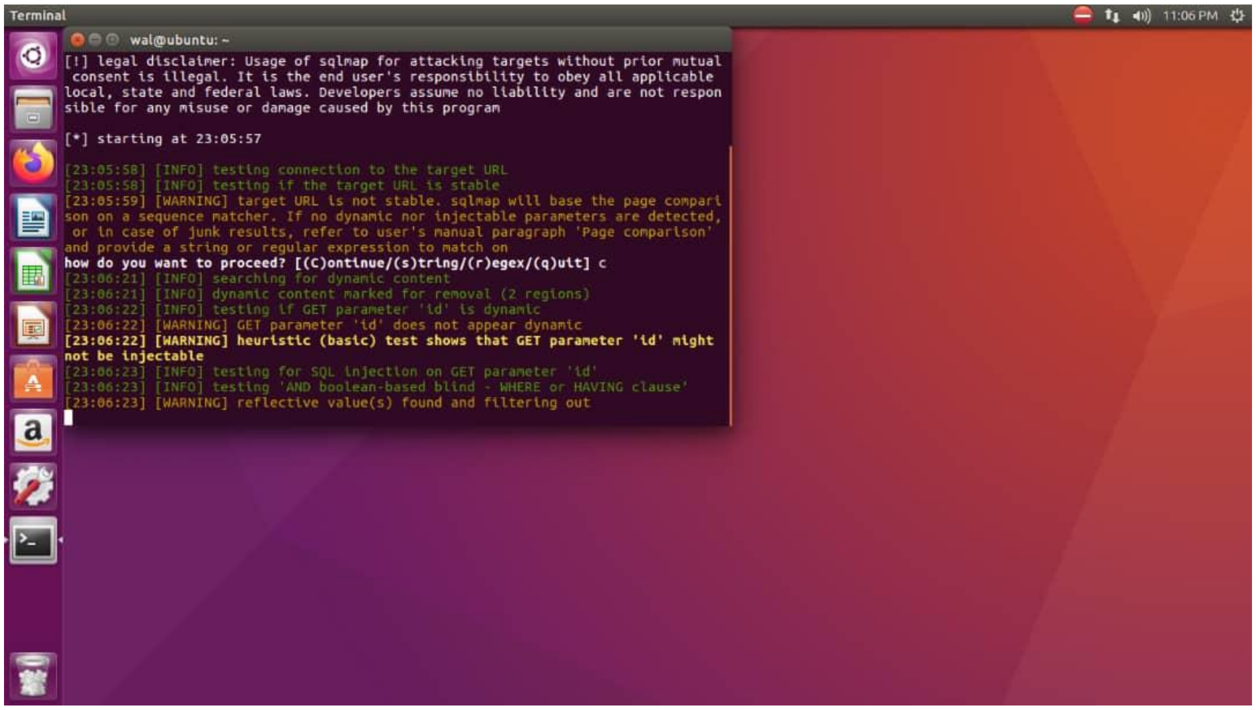
Table (4.1): comparing table

4.10 The Analysis:

By Linux Kali operating System and There Tools , Here Applied SQL Map Tool for Detection SQLIA.

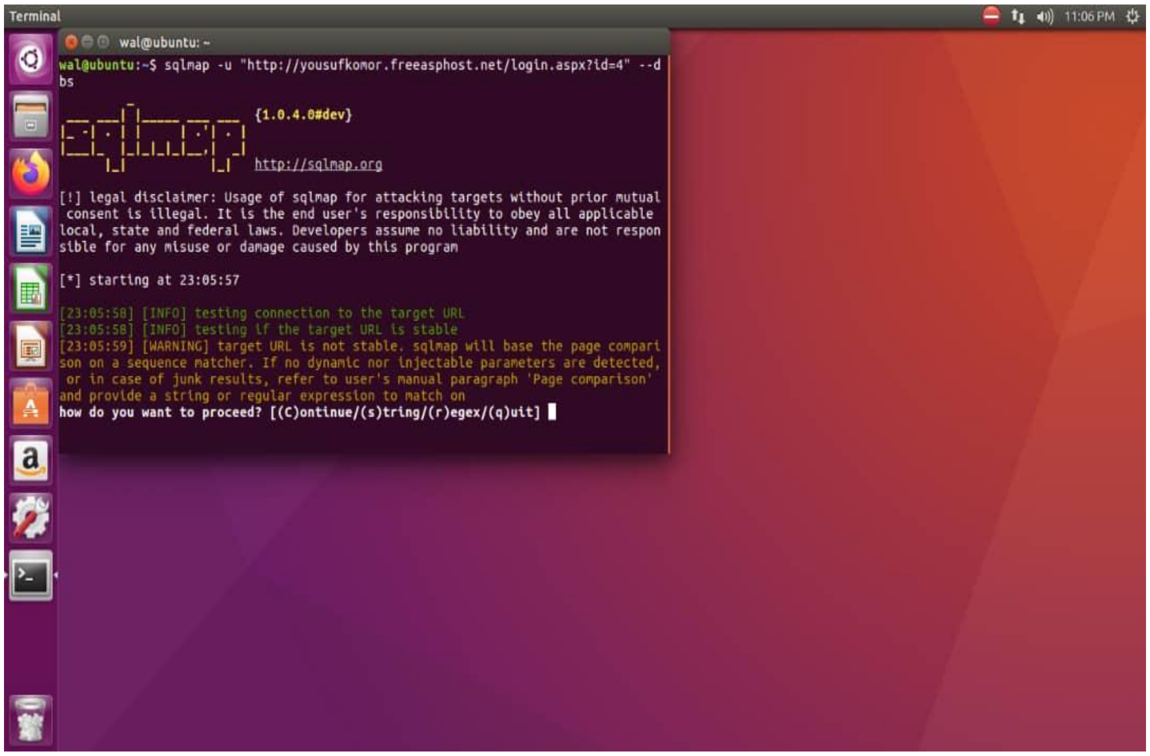
SQLMAP is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections [29]

Figure (4.17): Shows the starting of SQL MAP tool in Linux Khali



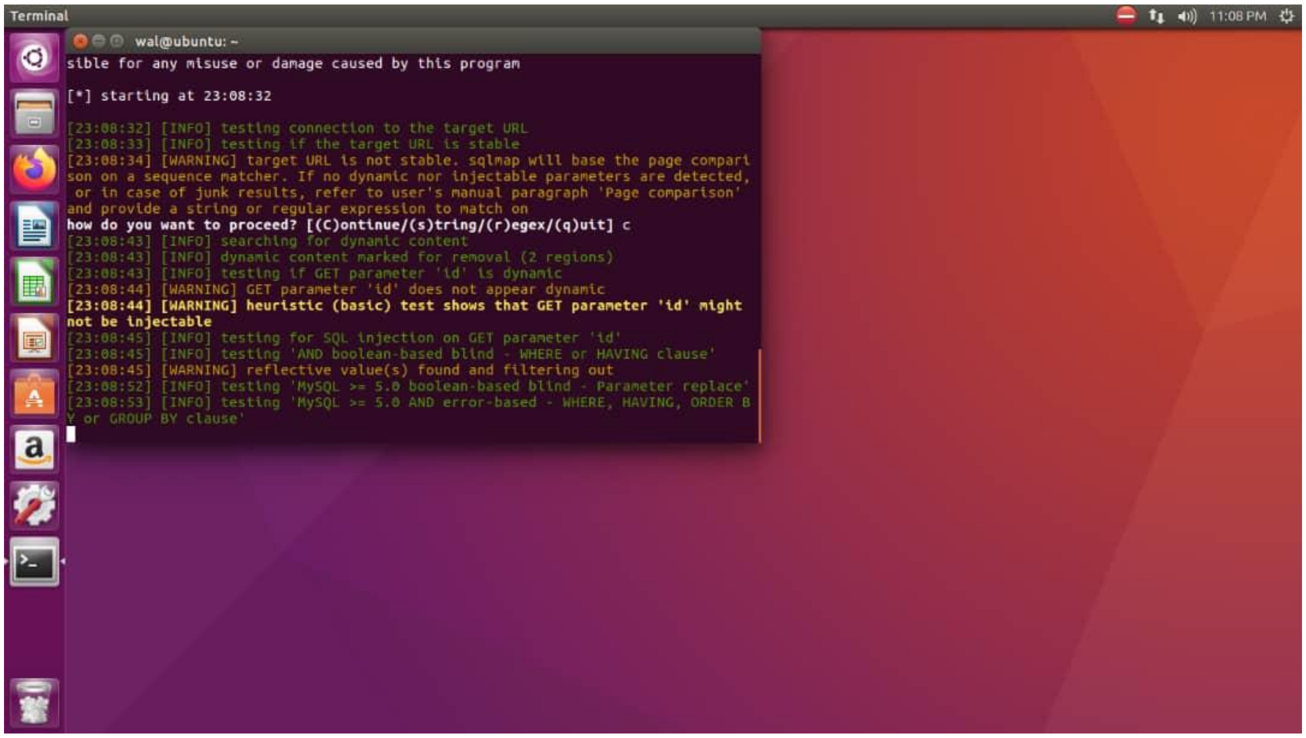
```
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
 consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 23:05:57

[23:05:58] [INFO] testing connection to the target URL
[23:05:58] [INFO] testing if the target URL is stable
[23:05:59] [WARNING] target URL is not stable. sqlmap will base the page compari
son on a sequence matcher. If no dynamic nor injectable parameters are detected,
 or in case of junk results, refer to user's manual paragraph 'Page comparison'
and provide a string or regular expression to match on
how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit] c
[23:06:21] [INFO] searching for dynamic content
[23:06:21] [INFO] dynamic content marked for removal (2 regions)
[23:06:22] [INFO] testing if GET parameter 'id' is dynamic
[23:06:22] [WARNING] GET parameter 'id' does not appear dynamic
[23:06:22] [WARNING] heuristic (basic) test shows that GET parameter 'id' might
not be injectable
[23:06:23] [INFO] testing for SQL injection on GET parameter 'id'
[23:06:23] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[23:06:23] [WARNING] reflective value(s) found and filtering out
```

Figure (4.17): Starting SQL Map WITH URL login page



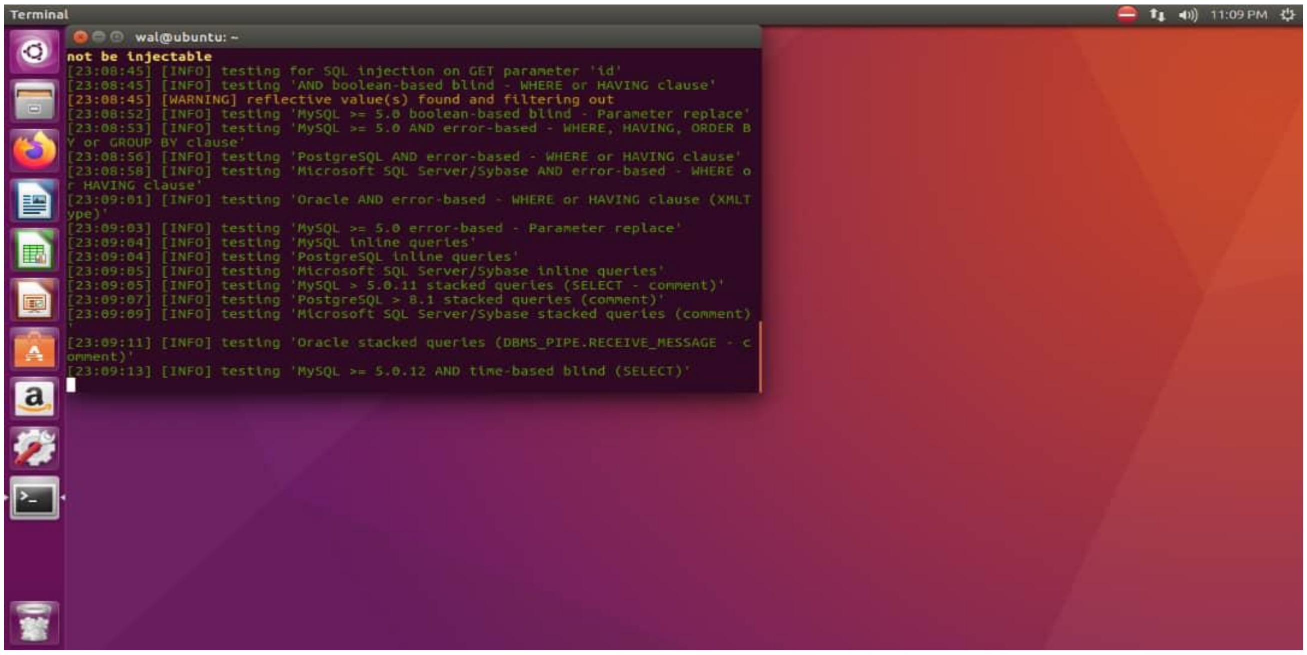
```
wal@ubuntu:~$ sqlmap -u "http://yousufkomor.freeasphost.net/login.aspx?id=4" --d
bs
                              {1.0.4.0#dev}

                              http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
 consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 23:05:57

[23:05:58] [INFO] testing connection to the target URL
[23:05:58] [INFO] testing if the target URL is stable
[23:05:59] [WARNING] target URL is not stable. sqlmap will base the page compari
son on a sequence matcher. If no dynamic nor injectable parameters are detected,
 or in case of junk results, refer to user's manual paragraph 'Page comparison'
and provide a string or regular expression to match on
how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit]
```

Figure (4.18): the process continue



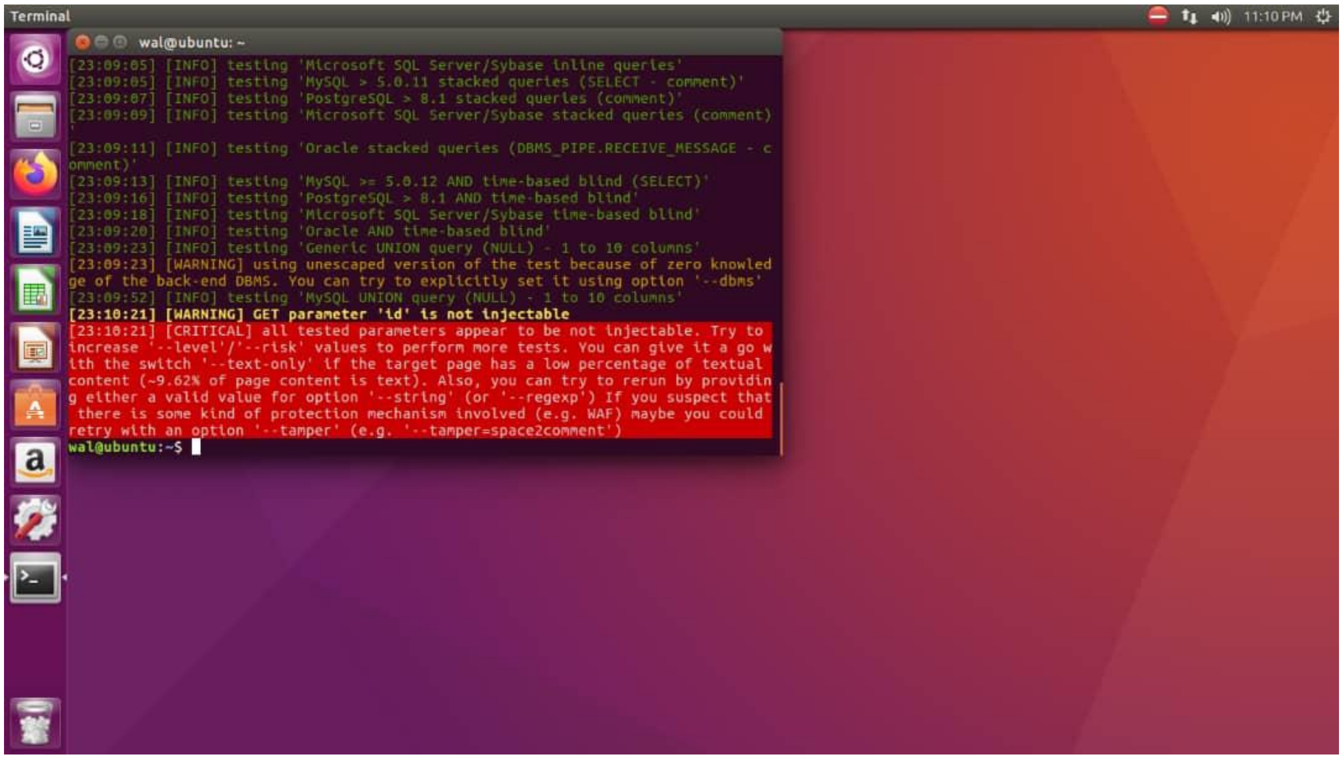
```
sible for any misuse or damage caused by this program

[*] starting at 23:08:32

[23:08:32] [INFO] testing connection to the target URL
[23:08:33] [INFO] testing if the target URL is stable
[23:08:34] [WARNING] target URL is not stable. sqlmap will base the page compari
son on a sequence matcher. If no dynamic nor injectable parameters are detected,
 or in case of junk results, refer to user's manual paragraph 'Page comparison'
and provide a string or regular expression to match on
how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit] c
[23:08:43] [INFO] searching for dynamic content
[23:08:43] [INFO] dynamic content marked for removal (2 regions)
[23:08:43] [INFO] testing if GET parameter 'id' is dynamic
[23:08:44] [WARNING] GET parameter 'id' does not appear dynamic
[23:08:44] [WARNING] heuristic (basic) test shows that GET parameter 'id' might
not be injectable
[23:08:45] [INFO] testing for SQL injection on GET parameter 'id'
[23:08:45] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[23:08:45] [WARNING] reflective value(s) found and filtering out
[23:08:52] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[23:08:53] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER B
Y or GROUP BY clause'
```

Figure (4.19): info for data base



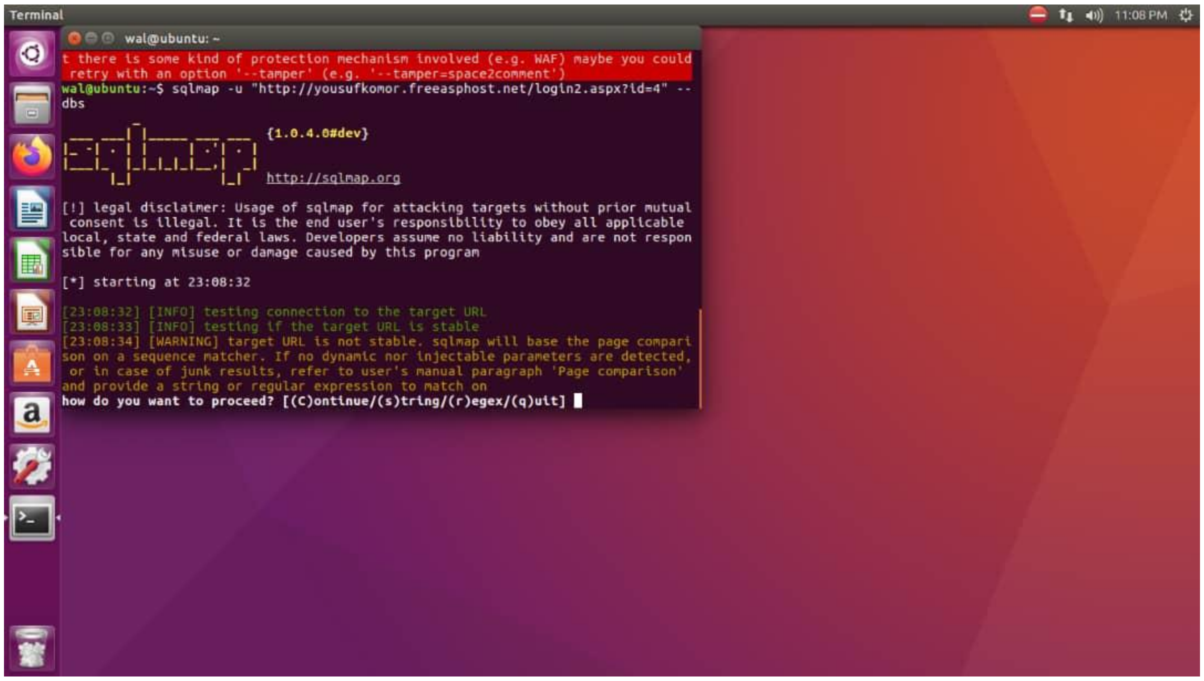
```
not be injectable
[23:08:45] [INFO] testing for SQL injection on GET parameter 'id'
[23:08:45] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[23:08:45] [WARNING] reflective value(s) found and filtering out
[23:08:52] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[23:08:53] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER B
Y or GROUP BY clause'
[23:08:56] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[23:08:58] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE o
r HAVING clause'
[23:09:01] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLT
ype)'
[23:09:03] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace'
[23:09:04] [INFO] testing 'MySQL inline queries'
[23:09:04] [INFO] testing 'PostgreSQL inline queries'
[23:09:05] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[23:09:05] [INFO] testing 'MySQL > 5.0.11 stacked queries (SELECT - comment)'
[23:09:07] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[23:09:09] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)
'
[23:09:11] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - c
omment)'
[23:09:13] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (SELECT)'
```

Figure (4.20): testing payloads



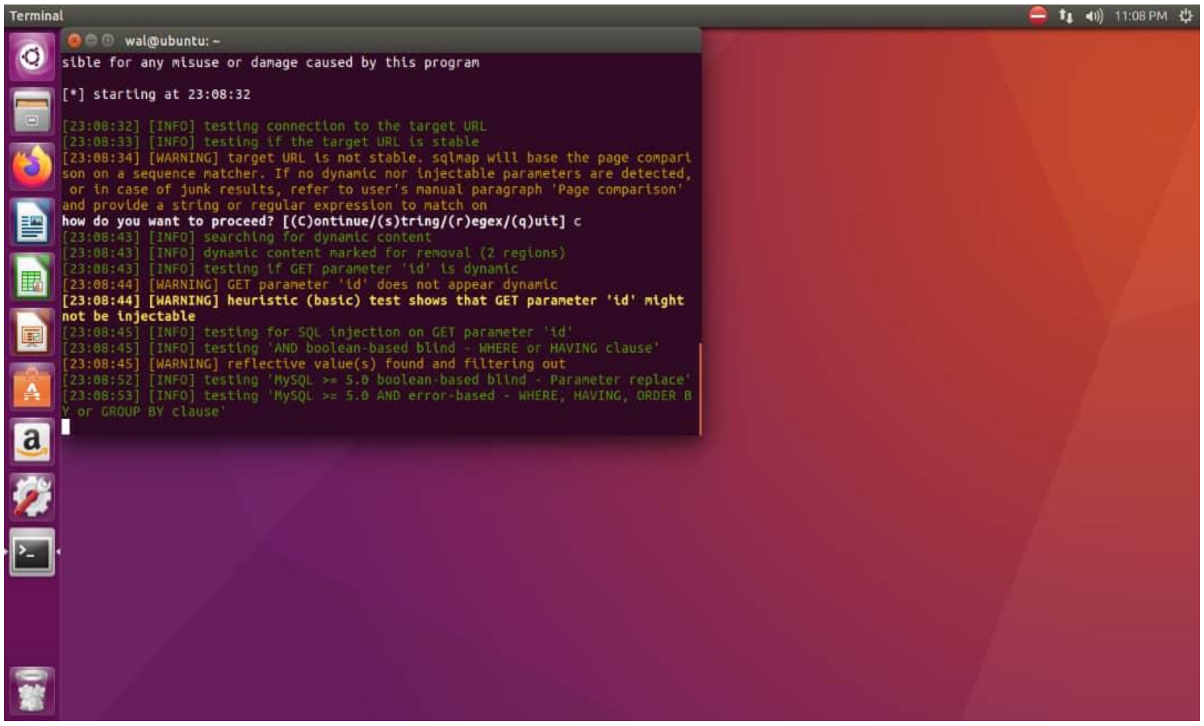
```
[23:09:05] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[23:09:05] [INFO] testing 'MySQL > 5.0.11 stacked queries (SELECT - comment)'
[23:09:07] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[23:09:09] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)
'
[23:09:11] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - c
omment)'
[23:09:13] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (SELECT)'
[23:09:16] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[23:09:18] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind'
[23:09:20] [INFO] testing 'Oracle AND time-based blind'
[23:09:23] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[23:09:23] [WARNING] using unescaped version of the test because of zero knowled
ge of the back-end DBMS. You can try to explicitly set it using option '--dbms'
[23:09:52] [INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'
[23:10:21] [WARNING] GET parameter 'id' is not injectable
[23:10:21] [CRITICAL] all tested parameters appear to be not injectable. Try to
increase '--level'/'--risk' values to perform more tests. You can give it a go w
ith the switch '--text-only' if the target page has a low percentage of textual
content (~9.62% of page content is text). Also, you can try to rerun by providin
g either a valid value for option '--string' (or '--regexp') If you suspect that
 there is some kind of protection mechanism involved (e.g. WAF) maybe you could
retry with an option '--tamper' (e.g. '--tamper=space2comment')
wal@ubuntu:~$
```

Figure (4.21): tested parameter



```
t there is some kind of protection mechanism involved (e.g. WAF) maybe you could
 retry with an option '--tamper' (e.g. '--tamper=space2comment')
wal@ubuntu:~$ sqlmap -u "http://yousufkomor.freeasphost.net/login2.aspx?id=4" --
dbs
        _
 ___ ___| |_____ ___ ___  {1.0.4.0#dev}
|_ -| . | |     | .'| . |
|___|_  |_|_|_|_|__,|  _|
      |_|           |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
 consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 23:08:32

[23:08:32] [INFO] testing connection to the target URL
[23:08:33] [INFO] testing if the target URL is stable
[23:08:34] [WARNING] target URL is not stable. sqlmap will base the page compari
son on a sequence matcher. If no dynamic nor injectable parameters are detected,
 or in case of junk results, refer to user's manual paragraph 'Page comparison'
and provide a string or regular expression to match on
how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit]
```

Figure (4.22): Starting the Analysis in URL login2

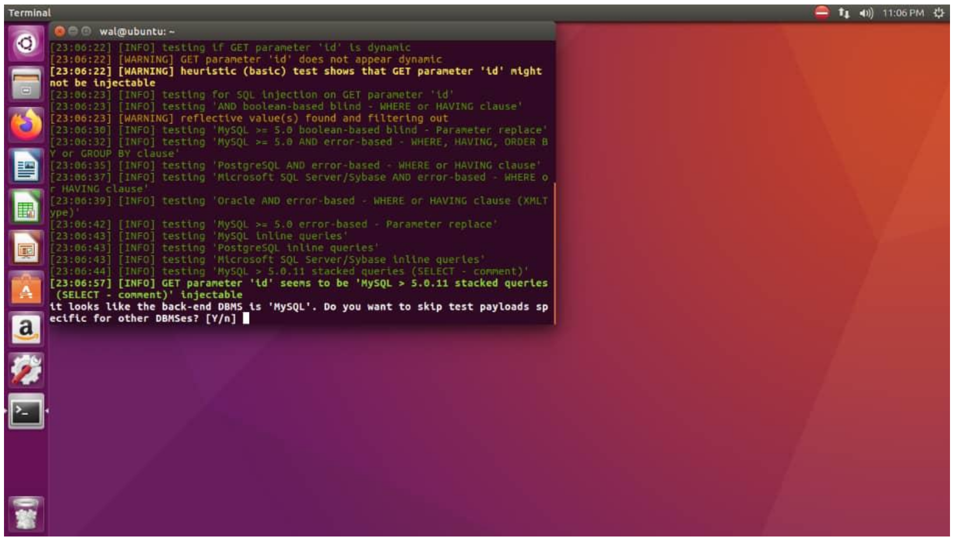


```
sible for any misuse or damage caused by this program

[*] starting at 23:08:32

[23:08:32] [INFO] testing connection to the target URL
[23:08:33] [INFO] testing if the target URL is stable
[23:08:34] [WARNING] target URL is not stable. sqlmap will base the page compari
son on a sequence matcher. If no dynamic nor injectable parameters are detected,
 or in case of junk results, refer to user's manual paragraph 'Page comparison'
and provide a string or regular expression to match on
how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit] c
[23:08:43] [INFO] searching for dynamic content
[23:08:43] [INFO] dynamic content marked for removal (2 regions)
[23:08:43] [INFO] testing if GET parameter 'id' is dynamic
[23:08:44] [WARNING] GET parameter 'id' does not appear dynamic
[23:08:44] [WARNING] heuristic (basic) test shows that GET parameter 'id' might
not be injectable
[23:08:45] [INFO] testing for SQL injection on GET parameter 'id'
[23:08:45] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[23:08:45] [WARNING] reflective value(s) found and filtering out
[23:08:52] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[23:08:53] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER B
Y or GROUP BY clause'
```

Figure (4.23): the process continue

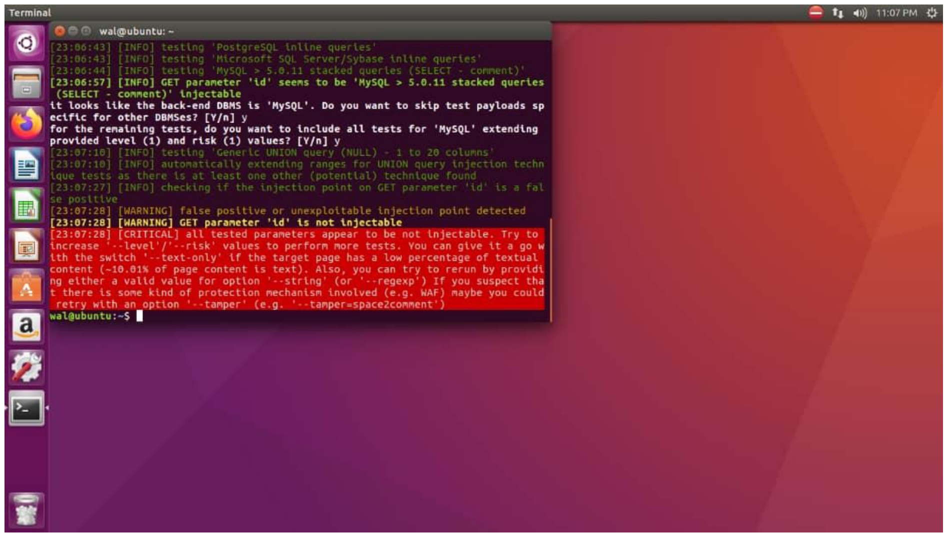
Figure (4.24) (4.25): the Analysis with SQLMAP detected by Three Types of SQLIA :

1. UNION query-based
2. Boolean- based blind
3. Error based



```
[23:06:22] [INFO] testing if GET parameter 'id' is dynamic
[23:06:22] [WARNING] GET parameter 'id' does not appear dynamic
[23:06:22] [WARNING] heuristic (basic) test shows that GET parameter 'id' might
not be injectable
[23:06:23] [INFO] testing for SQL injection on GET parameter 'id'
[23:06:23] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[23:06:23] [WARNING] reflective value(s) found and filtering out
[23:06:30] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'
[23:06:32] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER B
Y or GROUP BY clause'
[23:06:35] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[23:06:37] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE o
r HAVING clause'
[23:06:39] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLT
ype)'
[23:06:42] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace'
[23:06:43] [INFO] testing 'MySQL inline queries'
[23:06:43] [INFO] testing 'PostgreSQL inline queries'
[23:06:43] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[23:06:44] [INFO] testing 'MySQL > 5.0.11 stacked queries (SELECT - comment)'
[23:06:57] [INFO] GET parameter 'id' seems to be 'MySQL > 5.0.11 stacked queries
 (SELECT - comment)' injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads sp
ecific for other DBMSes? [Y/n]
```

Figure (4.24): detected three types of SQLIA



```
[23:06:43] [INFO] testing 'PostgreSQL inline queries'
[23:06:43] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[23:06:44] [INFO] testing 'MySQL > 5.0.11 stacked queries (SELECT - comment)'
[23:06:57] [INFO] GET parameter 'id' seems to be 'MySQL > 5.0.11 stacked queries
 (SELECT - comment)' injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads sp
ecific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL' extending
provided level (1) and risk (1) values? [Y/n] y
[23:07:10] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[23:07:10] [INFO] automatically extending ranges for UNION query injection techn
ique tests as there is at least one other (potential) technique found
[23:07:27] [INFO] checking if the injection point on GET parameter 'id' is a fal
se positive
[23:07:28] [WARNING] false positive or unexploitable injection point detected
[23:07:28] [WARNING] GET parameter 'id' is not injectable
[23:07:28] [CRITICAL] all tested parameters appear to be not injectable. Try to
increase '--level'/'--risk' values to perform more tests. You can give it a go w
ith the switch '--text-only' if the target page has a low percentage of textual
content (~10.01% of page content is text). Also, you can try to rerun by providi
ng either a valid value for option '--string' (or '--regexp') If you suspect tha
t there is some kind of protection mechanism involved (e.g. WAF) maybe you could
 retry with an option '--tamper' (e.g. '--tamper=space2comment')
wal@ubuntu:~$
```

Figure (4.25): Info for database (MY SQL)

Here the sql map founded the database of my web site and information about it .

4. 11 Results:

According to the analysis and practical part the flowing results are gutted:

1. There are numbers of methods to detect the SQL injection which are Header by using (Get, Post function), textbox and payload.
2. Each method flows specific procedure to applying.
3. There are three main vulnerabilities which are SQL Injection, XSS and client-side attack as common vulnerabilities. To prevent from that risk there, have numbers of policies may be applying which are: Using Prepared Statements (with Parameterized Queries), Validating user input.
4. Any registration form not acts as active vulnerability.
5. Hacking May done on header or on entry textbox, but not done through check box or radon bottom or on select menus.
6. SQLMAP tool is best for detected SQL injection attack rather than Commix and Web Scan.

Chapter Five

Conclusion and Recommendation

5.1 Conclusion:

SQL injection attack is a code injection technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed. It is an instance of a more general class of vulnerabilities that can occur whenever one programming or scripting language is embedded inside another, in this research proposed method for detection and prevention of SQL attack is introduced the problem of this research acts in the SQL injection detection and prevention, the problem is one common issue of internet security and system, which begin at starting design web site. The research did an overview of the problem solution techniques applying two methods of prevention and other for detection SQLIA and tests the prevented system normally , The proposed model based on dynamic analyzer that work as four phases are Planning Phase, Discovery Phase, Attack phase , Reporting phase . The results of the research the system secure against SQL injection attack .

5.2 Recommendation:

After practical part and interpreting the results I recommend by the flowing:

1. Use the exceptions handler function to perform the prevention.
2. Suggest new methods to execute SQL Command.
3. Full fill of registration form or similar of it to prevent this type of attacks.

References:

[1] http:// www.cgi security.com access time 10:00 pm -14.2.2021

[2] http:// www.Zone -h.org/archive . com access time 1:00 am -1.2.2021

[3] http:// www.security doc.com/library/2651 access time 1:15 am -15.2.2021

[4] http:// www.Comsecurity concepts.wordpress.com access time 9:00 am -5.3.2021

[5] http:// www. Sec lists .org/ bugtraq /2020/ Feb/0288.html access time 2:20 pm -10.3.2021

[6] http://www. Phoenixnap.com access time 9:00 pm -10.3.2021

[7] http:// www.imperva.com/resources/whiteapers access time 6:00 am -11.3.2021

[8] http:// www. Researchgate.net .com access time 9:00 am -11.3.2021

[9] http:// www.wikipedia.com access time 4:00 pm -15.3.2021

[10] http:// www.Techopedia.com access time 6:00 am -1.4.2021

[11] http:// www.net-security.org /d1/article/blind-SQL injection.pdf

Access time 10:00 am -1.4.2021

[12] http:// www.OWASP.com /blind SQL injection-OWASP.html

Access time 6:00 am -4.4.2021

[13] http:// www. link.springer.com access time 7:00 am - 4.4.2021

[14] http:// www.help net security.com access time 8:00 am - 4.4.2021

[15] Rania Alsahafi , “SQL Injection Attacks: Detection And Prevention Techniques” International Journal Of Scientific & Technology Research , JANUARY 2019

[16] http:// www. Spanning.com access time 6:00 am -15.4.2021

[17] http:// www. Issue .com access time 6:00 am -20.4.2021

[18] http:// www.cloudacademy.com access time 8:00 am -25.4.2021

[19] http:// www. github .com access time 9:00 am -25.4.2021

[20] http:// www. cyper-today.com access time 11:00 am -27.4.2021

[21]   http:// www. Linux security.expert.com   access time 12:00 am -27.4.2021

[22] https://doi.org/10.21203/rs.3.rs-1321852/v1

[23] F. G.Deriba , T.M.Kassa , W.B.Demi lie ,” Attacks on SQL Injection and Developing Compressive Framework Using a Hybrid and Machine Learning Approach” , vol. 12, no. 6, pp.324–332, 2022.

[24] D. Chen, Q. Yan, C. Wu, J. Zhao,” SQL Injection Attack Detection and Prevention Techniques Using Deep Learning”, 2021 J. Phys.: Conf. Ser. 1757 012055, doi:10.1088/1742-6596/1757/1/012055

[25] K. Ahmad , M. Karim ,” A Method to Prevent SQL Injection Attack using an Improved Parameterized Stored Procedure” , International Journal of Computer Applications, Vol. 12, No. 6, 2021

[26] O. C. Abikoye, A. Abubakar, A. H.Dokoro , O. N. Akande and A. A Kayode , “A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm” , Journal on Information Security (2020) 2020:14 https://doi.org/10.1186/s13635-020-00113-y

[27] Rania Alsahafi ,” SQL Injection Attacks: Detection And Prevention Techniques” , International Journal of scientific & Technology Research Vol .8 , ISSUE 01, JANUARY 2019 ISSN 2277-8616

[28] M. A. Azman, M. F. Marhusin, R. Sulaiman, U. Sains, M. F. Marhusin, and U. Sains, “Machine Learning-Based Technique to Detect SQL Injection Attack,” pp. 1–8, 2021, DOI: 10.3844/jcssp.2021.296.303.

[29]   http:// www. sqlmap .org   access time 5:00 pm -1.5.2021