



Ethical Hacking Essentials (EHE)

1 Master of Science in Cybersecurity



Dr. Kamal Aliyan

Everyone Smart University

Faculty of IT – 2022/2023

Web Application Attacks and Countermeasures

Module Objectives

- 1 Understanding Web Server Concepts and Attacks
- 2 Understanding Different Web Server Attack Tools and Countermeasures
- 3 Overview of Web Application Architecture and Vulnerability Stack
- 4 Understanding Different Web Application Threats and Attacks
- 5 Understanding Different Web Application Attack Tools and Countermeasures
- 6 Overview of Different Types of SQL Injection Attacks
- 7 Understanding Different SQL Injection Tools
- 8 Understanding Different SQL Injection Attack Countermeasures

SQL Injection Attacks

Module Flow

1

**Discuss Types of SQL
Injection Attacks**

2

**Discuss SQL Injection Attack
Countermeasures**

What is SQL Injection?

- ❑ SQL injection is a technique used to take advantage of **un-sanitized input vulnerabilities** to pass SQL commands through a web application for execution by a **backend database**
- ❑ It is a basic attack used to either **gain unauthorized access** to a database or **retrieve information** directly from the database



Why Bother about SQL Injection?



Based on the use of **applications** and the way they **process user supplied data**, SQL injections can be used to implement the following types of attacks:

1 Authentication Bypass

Compromised Data Integrity **4**

2 Authorization Bypass

Compromised Availability of Data **5**

3 Information Disclosure

Remote Code Execution **6**

SQL Injection and Server-side Technologies



Server-side Technology

Powerful server-side technologies like ASP.NET and database servers allow developers to **create dynamic, data-driven websites**, and **web apps** with incredible ease



Exploit

The power of ASP.NET and SQL can easily be **exploited by hackers** using SQL injection attacks



Susceptible Databases

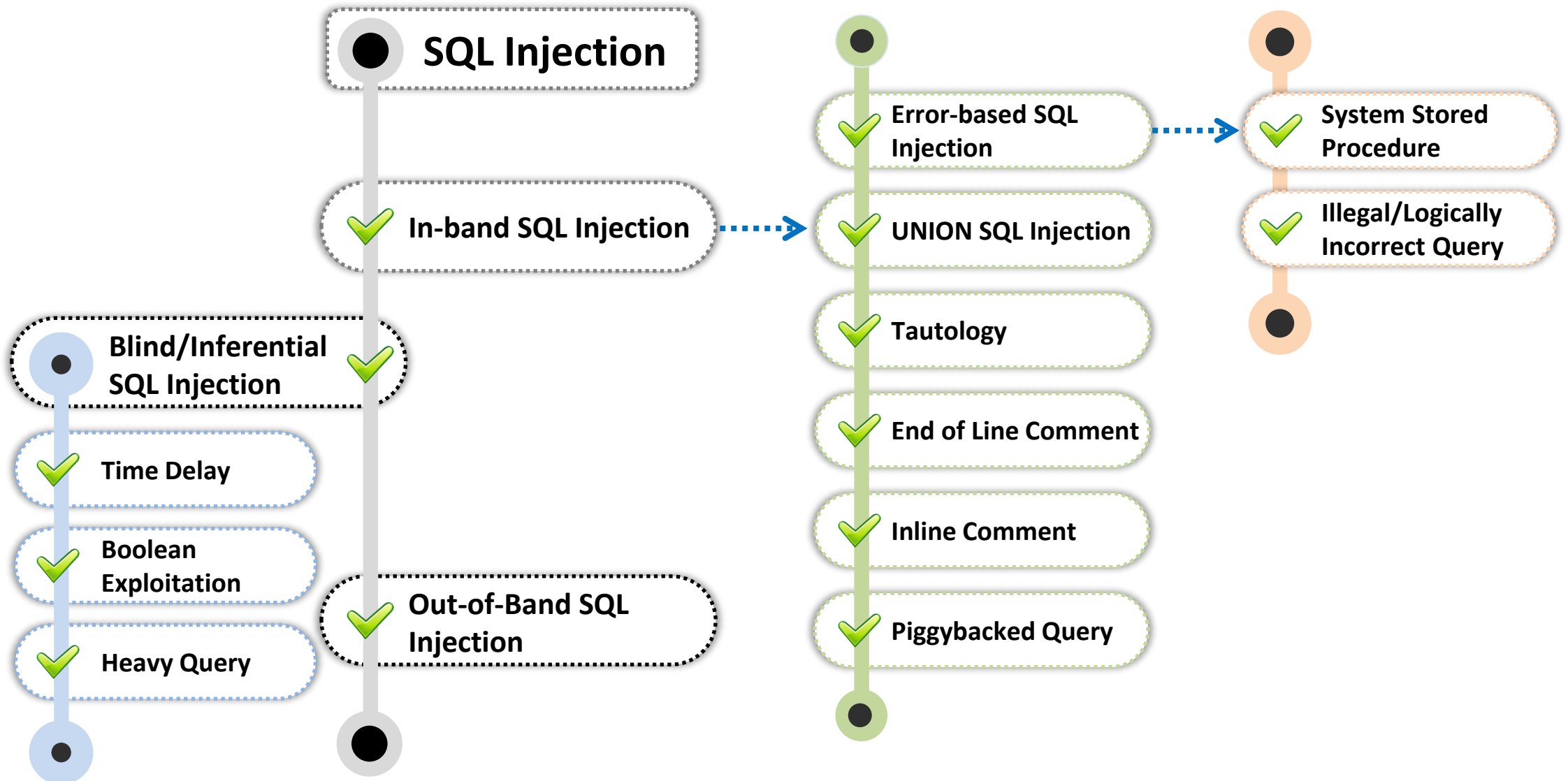
All relational databases, SQL Server, Oracle, IBM DB2, and MySQL, are susceptible to **SQL-injection attacks**



Attack

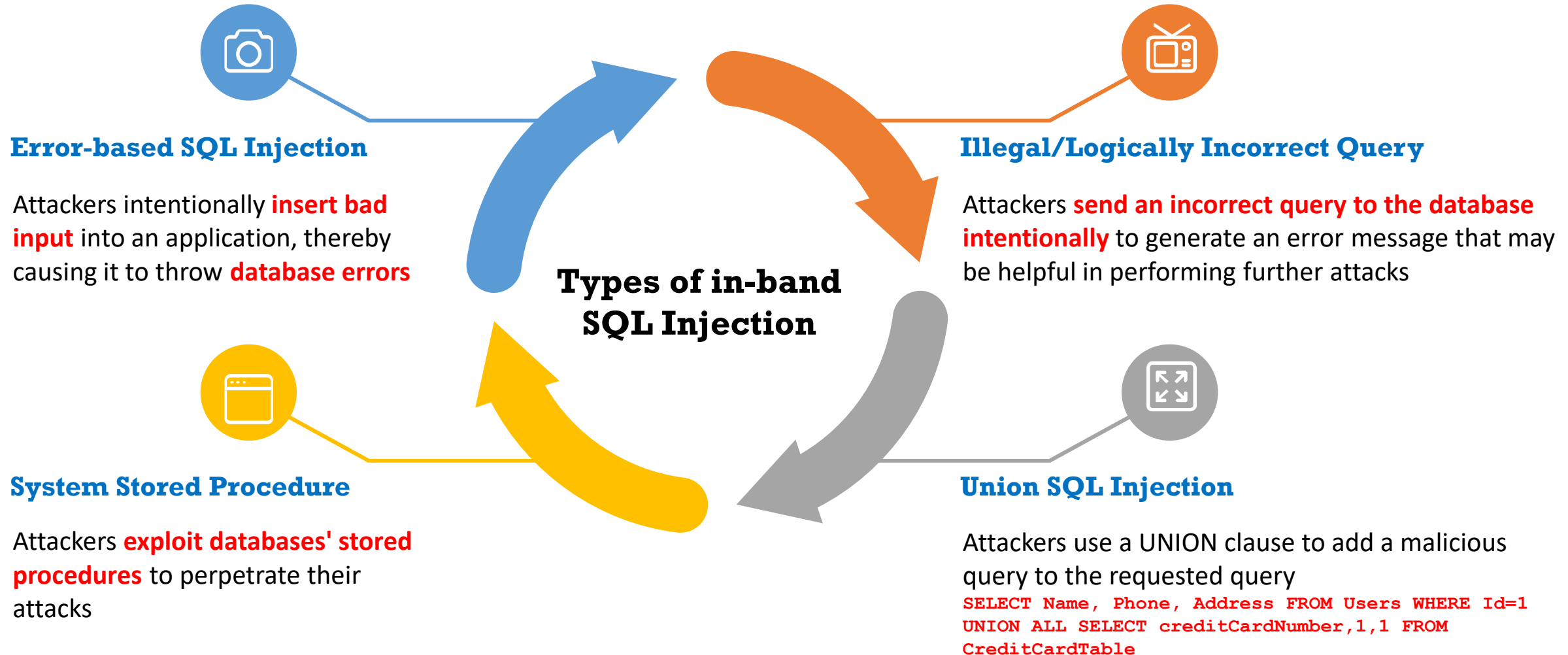
SQL injection attacks do not exploit a specific software vulnerability, instead they **target websites and web apps** that do not follow **secure coding practices** for accessing and manipulating data stored in a relational database

Types of SQL Injection



In-Band SQL Injection

- ❑ Attackers use the **same communication channel** to perform the attack and **retrieve** the results



In-Band SQL Injection (Cont'd)

Types of in-band SQL Injection

Tautology

Attackers inject statements that are always true so that the queries always return results after evaluating the WHERE condition

```
SELECT * FROM users WHERE name =  
    '' OR '1'='1';
```

End of Line Comment

After injecting the code into a specific field, legitimate code that follows is nullified using end of line comments

```
SELECT * FROM user WHERE name =  
    'x' AND userid IS NULL; --';
```



In-line Comments

Attackers integrate multiple vulnerable inputs into a single query using inline comments

```
INSERT INTO Users (UserName,  
isAdmin, Password)  
VALUES('Attacker', 1, /*', 0,  
    '*/'mypwd')
```

Piggybacked Query

Attackers inject additional malicious query into the original query. Consequently, the DBMS executes multiple SQL queries

```
SELECT * FROM EMP WHERE EMP.EID =  
1001 AND EMP.ENAME = 'Bob'; DROP  
TABLE DEPT;
```

Error Based SQL Injection



- ❑ Error based SQL Injection **forces the database** to perform some operation in which the **result will be an error**
- ❑ This exploitation may differ depending on the DBMS

- ✓ Consider the SQL query shown below:

```
SELECT * FROM products WHERE  
id_product=$id_product
```

- ✓ Consider the following request to a script that executes the query above:

```
http://www.example.com/product.php?id=10
```

- ✓ The malicious request would be (e.g., Oracle 10g):

```
http://www.example.com/product.php?  
id=10||UTL_INADDR.GET_HOST_NAME( (SELECT user  
FROM DUAL) )-
```



Union SQL Injection



- ❑ This technique involves **joining a forged query** to the **original query**
- ❑ The result of a forged query will be joined to the result of the original query, thereby allowing it to obtain the **values of fields of other tables**

Example:

• `SELECT Name, Phone, Address FROM Users WHERE Id=$id`

Now set the following Id value:

• `$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable`

The final query is as shown below:

• `SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable`

The above query joins the result of the original query with all the credit card users

Blind/Inferential SQL Injection



No Error Message

Blind SQL Injection is used when a **web application is vulnerable** to an SQL injection, but the results of the injection are not visible to the attacker



Generic Page

Blind SQL injection is identical to a normal SQL Injection, except that a generic custom page is displayed when an attacker attempts to exploit an application rather than seeing a **useful error message**

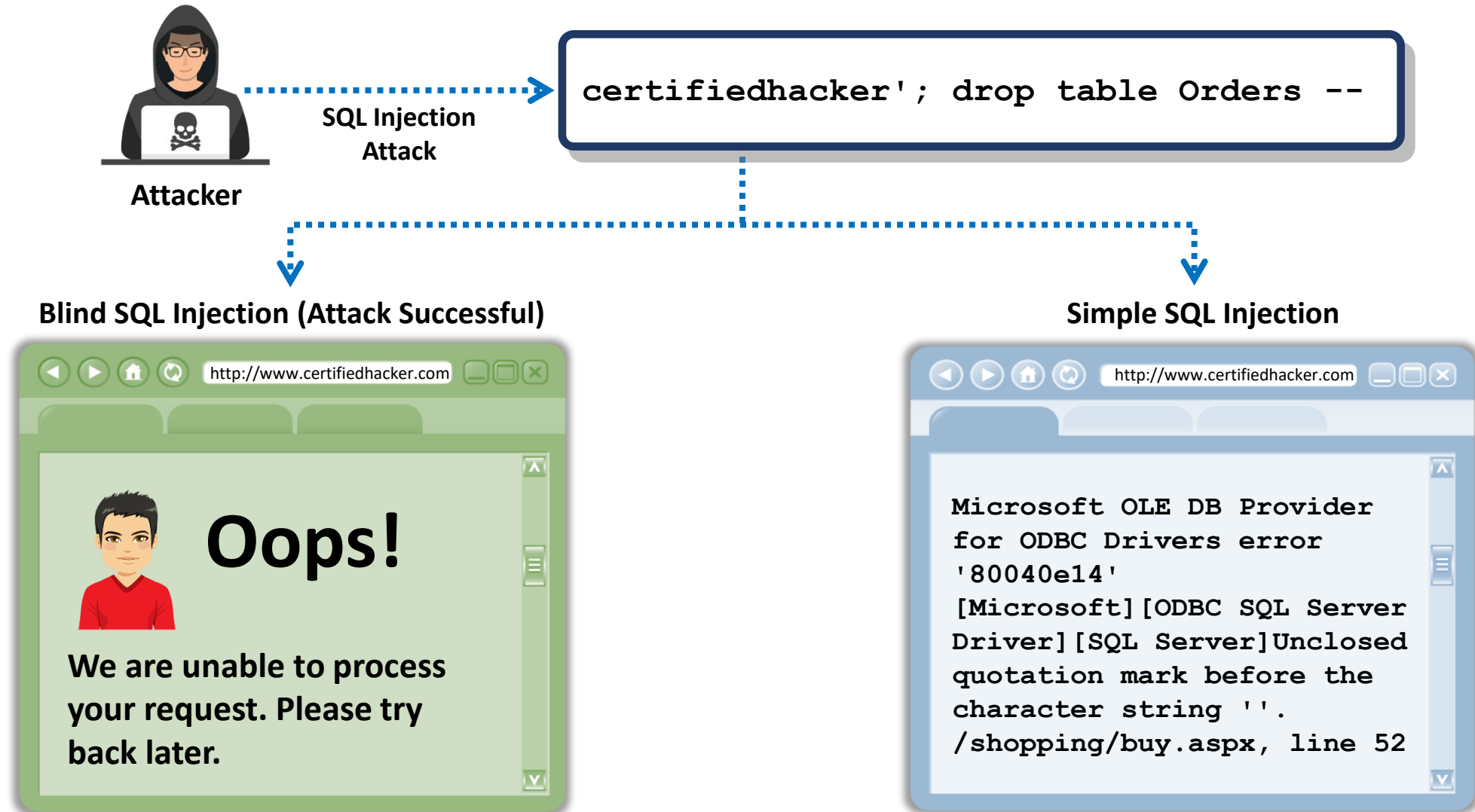


Time- intensive

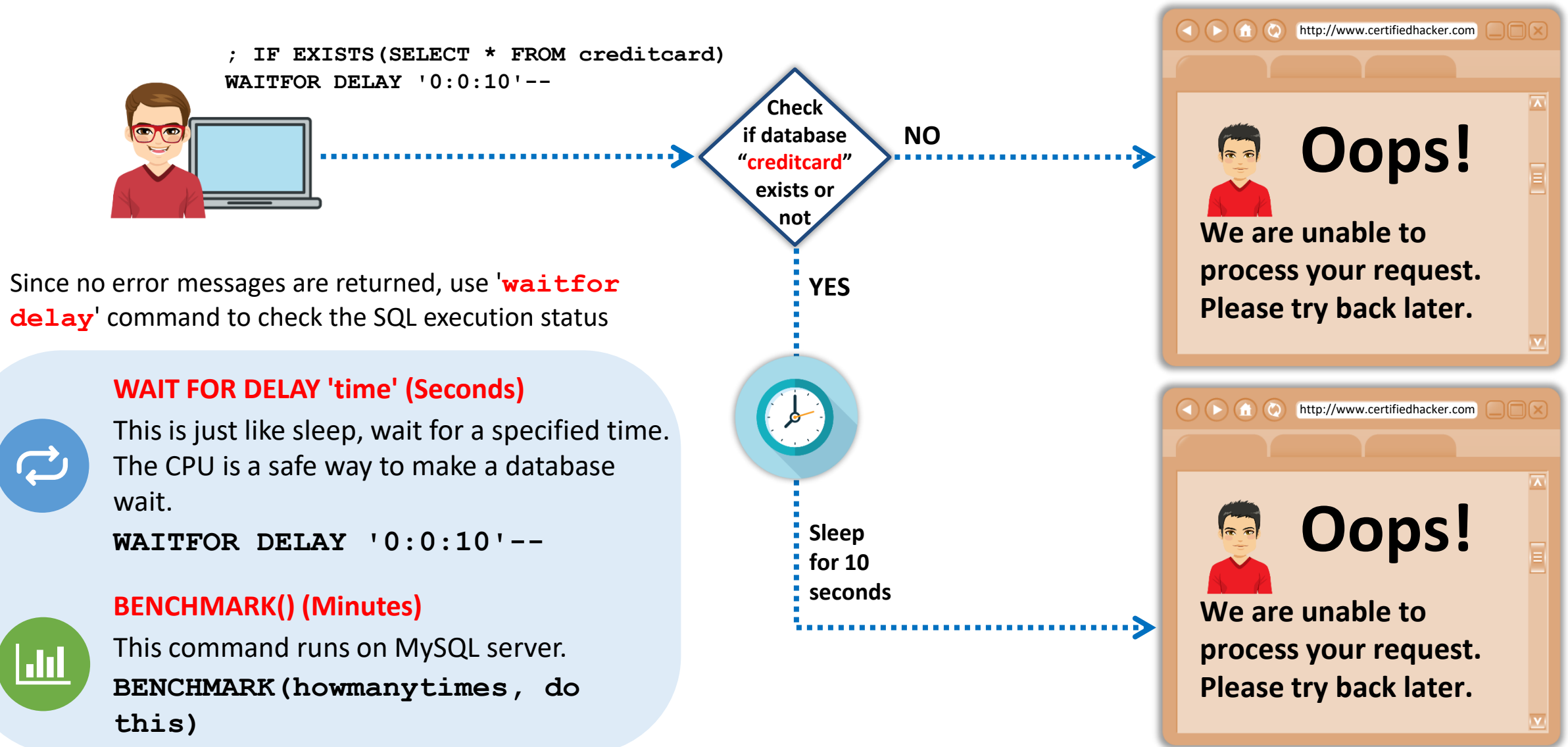
This type of attack can become **time-intensive because a new statement** must be crafted for each bit recovered

Note: An attacker can still steal data by asking a series of True and False questions through SQL statements

Blind SQL Injection: No Error Message Returned



Blind SQL Injection: WAITFOR DELAY (YES or NO Response)



Blind SQL Injection: Boolean Exploitation



Multiple valid statements that evaluate **true** and **false** are supplied in the affected parameter in the **HTTP request**



By comparing the response page between both conditions, the attackers can infer whether or not the **injection was successful**



For example, consider the following URL:

`http://www.myshop.com/item.aspx?id=67`

An attacker may manipulate the above request to

`http://www.myshop.com/item.aspx?id=67 and 1=2`

SQL Query Executed

```
SELECT Name, Price, Description FROM ITEM_DATA WHERE ITEM_ID = 67  
AND 1 = 2
```

Blind SQL Injection: Heavy Query

Attackers use heavy queries to perform a time delay SQL injection attack without using **time delay functions**

A heavy query retrieves a significant amount of data and takes a long time to execute in the **database engine**

Attackers generate heavy queries using **multiple joins on system tables**

For example,

```
SELECT * FROM products WHERE id=1 AND 1 <  
SELECT count(*) FROM all_users A,  
all_users B, all_users C
```



Out-of-Band SQL Injection

01

In Out-of-Band SQL injection, the attacker needs to **communicate with the server** and acquire features of the **database server** used by the web application

02

Attackers use different **communication channels** to perform the attack and obtain the results



03

Attackers use **DNS** and **HTTP requests** to retrieve data from the database server

04

For example, in a Microsoft SQL Server, an attacker exploits the **xp_dirtree** **command** to send DNS requests to a server controlled by the attacker

sqlmap

sqlmap



sqlmap



Mole



Blisqy



blind-sql-bitshifting



NoSQLMap



SQL Power Injector

Module Flow



1

**Discuss Types of SQL
Injection Attacks**



2

**Discuss SQL Injection Attack
Countermeasures**

SQL Injection Attack Countermeasures

1

Make no assumptions about the **size**, **type**, or **content** of the data that is received by your application

2

Test the **size** and **data type of input** and enforce appropriate limits to prevent buffer overruns

3

Test the content of **string variables** and accept only **expected values**

4

Reject entries that contain **binary data**, **escape sequences**, and **comment** characters

5

Never build **Transact-SQL** statements directly from user input and use stored procedures to validate user input

6

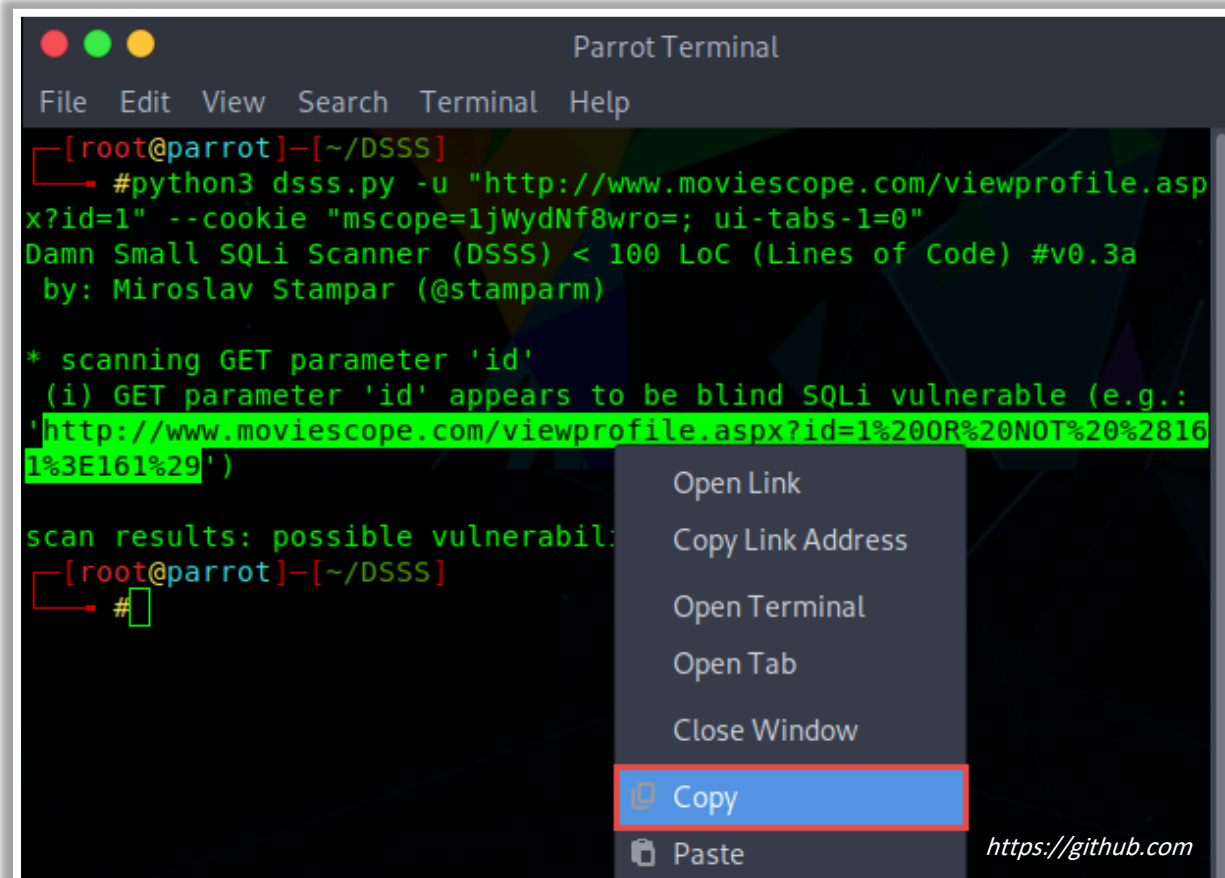
Implement **multiple layers of validation** and never concatenate user input that is not validated



SQL Injection Detection Tools

Damn Small SQLi Scanner (DSSS)

DSSS is an **SQL injection vulnerability scanner** that scans the web application for various SQL injection vulnerabilities



```
Parrot Terminal
File Edit View Search Terminal Help
[root@parrot]--[~/DSSS]
#python3 dsss.py -u "http://www.moviescope.com/viewprofile.aspx?id=1" --cookie "mscope=1jWydNf8wro=; ui-tabs-1=0"
Damn Small SQLi Scanner (DSSS) < 100 LoC (Lines of Code) #v0.3a
by: Miroslav Stampar (@stamparm)

* scanning GET parameter 'id'
(i) GET parameter 'id' appears to be blind SQLi vulnerable (e.g.:
'http://www.moviescope.com/viewprofile.aspx?id=1%20OR%20NOT%20%2816
1%3E161%29')
scan results: possible vulnerability
[root@parrot]--[~/DSSS]
#
```



OWASP ZAP

<https://www.owasp.org>



Snort

<https://www.snort.org>



Burp Suite

<https://portswigger.net>



HCL AppScan

<https://www.hcltech.com>



w3af

<https://w3af.org>

Module Summary



This module has discussed web server concepts and attacks



It has covered various web server attack tools and countermeasures



It discussed web application architecture and vulnerability stack



It also discussed various web application threats and attacks



It demonstrated different web application attack tools



It discussed various countermeasures against web application attacks



This module also discussed different types of SQL injection attacks and SQL injection tools



Finally, this module ended with a detailed discussion on various countermeasures against SQL injection attacks



In the next module, we will discuss in detail on various wireless attacks and countermeasures

