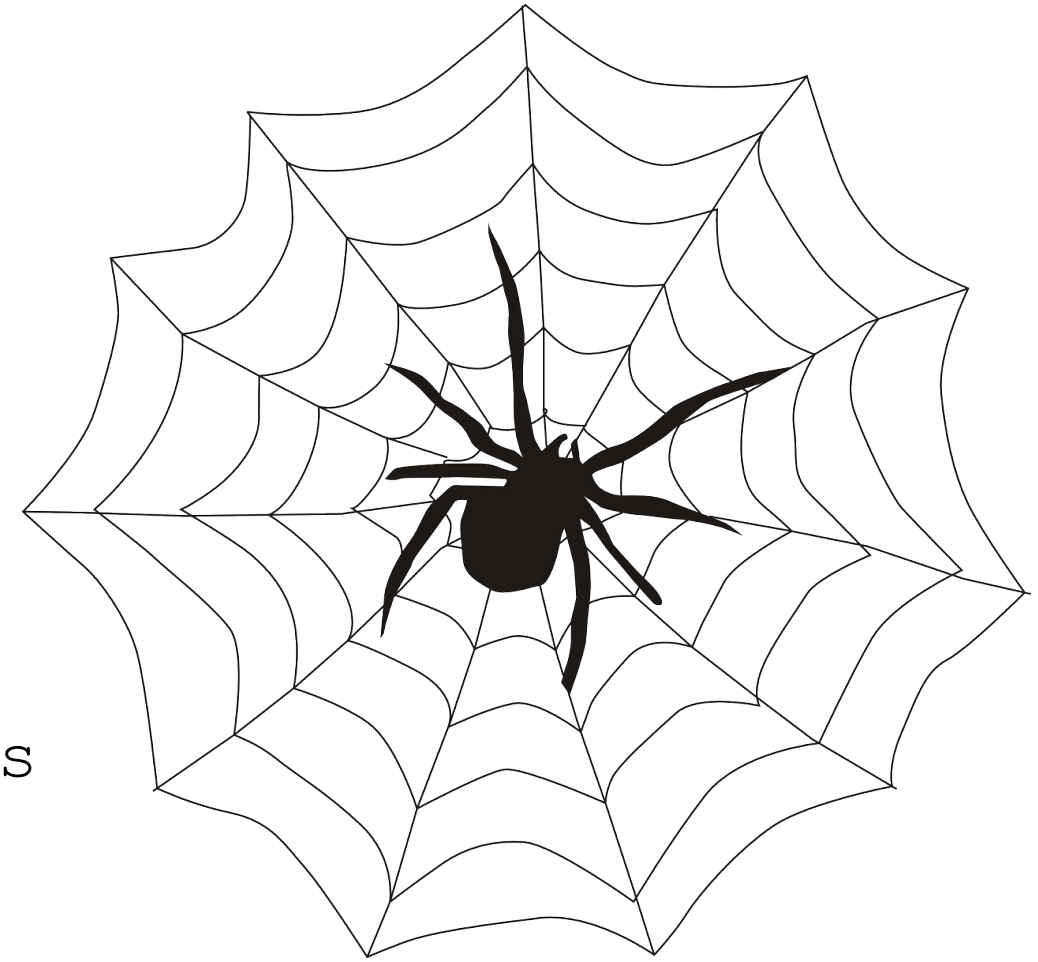
HackerFrogs Afterschool SQL Injection /w TryHackMe

Class:
Web App Hacking

Workshop Number:
AS-WEB-06

Document Version:
1.2

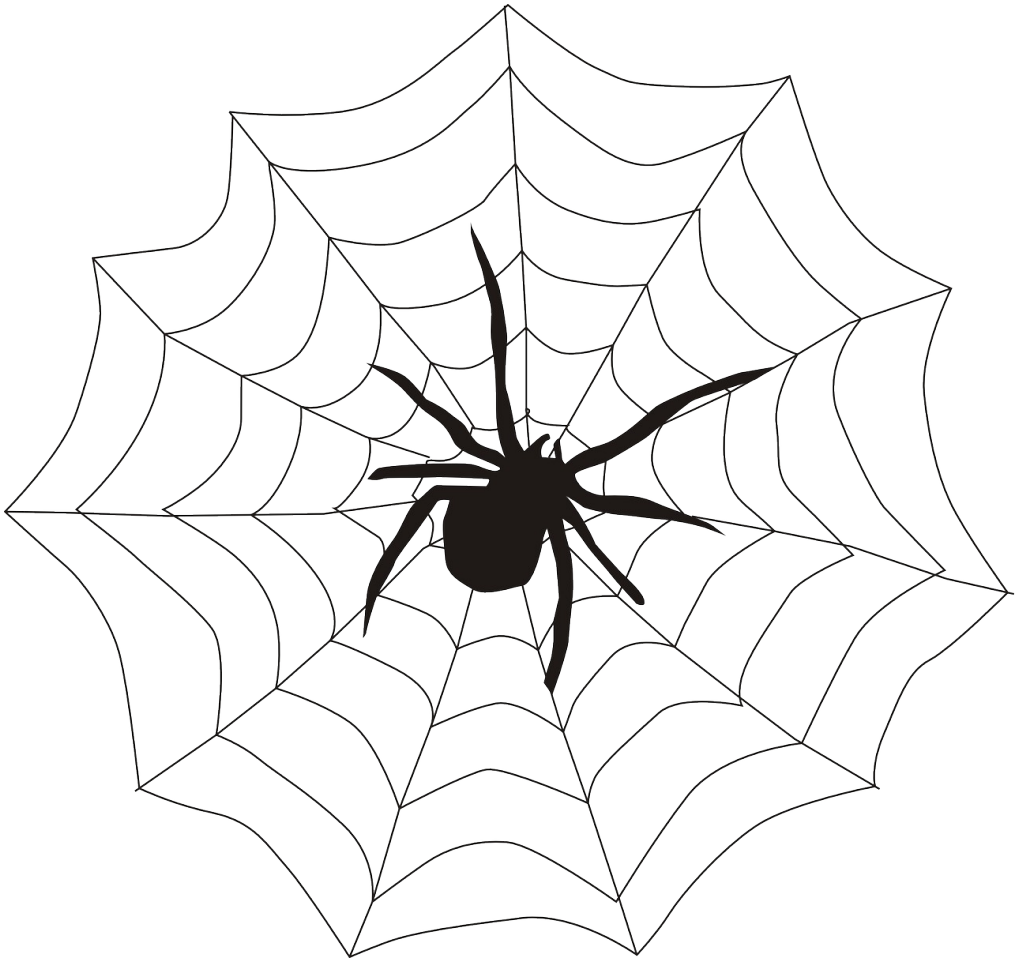
Special Requirements:
Completion of previous workshop, AS-WEB-05.
Registered account at tryhackme.com



What We Learned In The Previous Workshop

This is the sixth intro to web app hacking workshop.

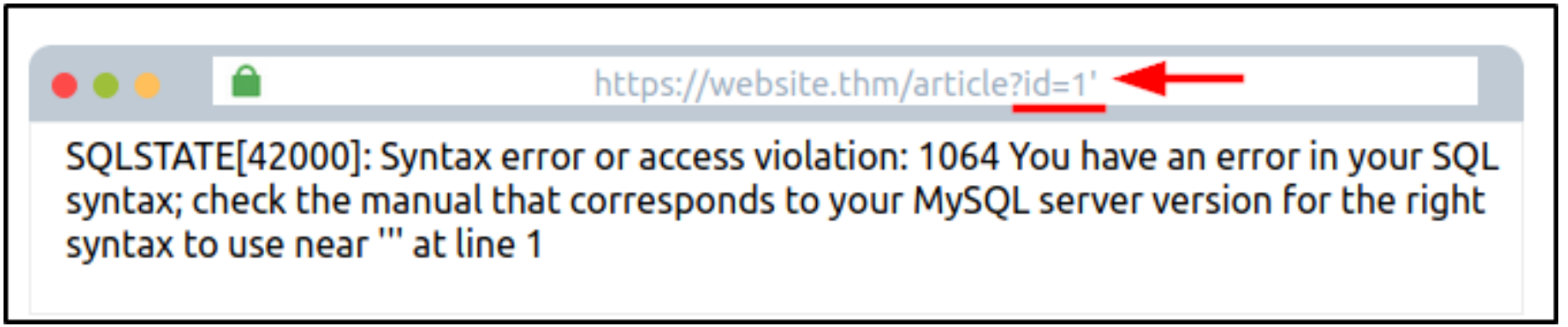
In the previous workshop we learned about the following web app hacking concepts:



SQL Injection

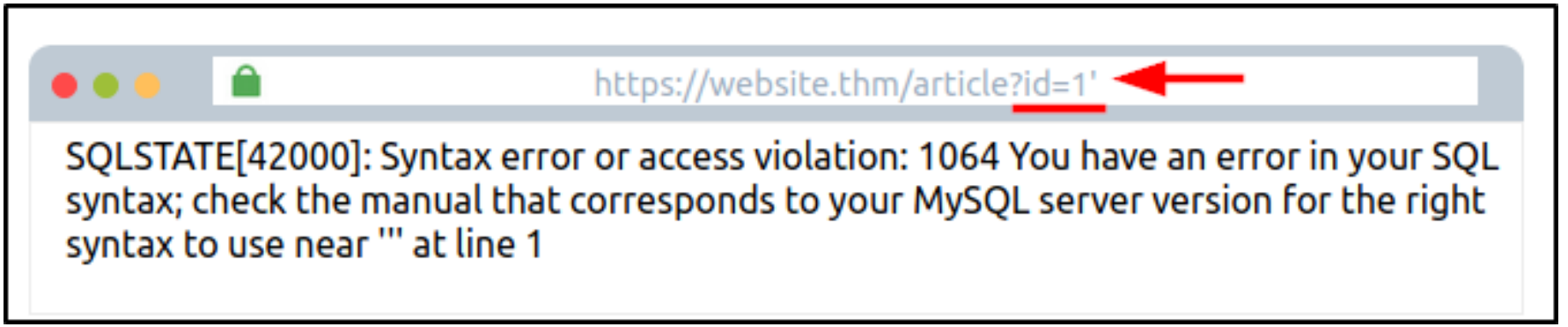
SQL Injection vulnerabilities can lead to sensitive data exposure, admin account takeover, webserver takeover, and more.

In-Band SQL Injection



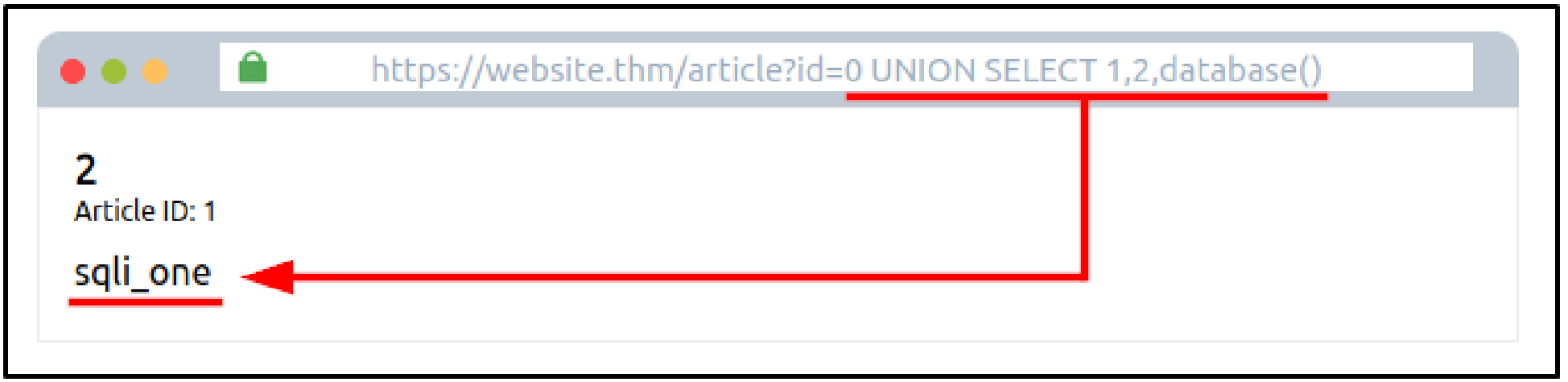
In-Band SQL Injection is SQL injection where the results of the injected query can be seen in the output of the web app. It is considered the easiest type of SQL injection to exploit.

Error-Based SQL Injection



Error-Based SQL Injection is a type of In-Band SQL Injection where error messages from the database software are returned to the web app interface.

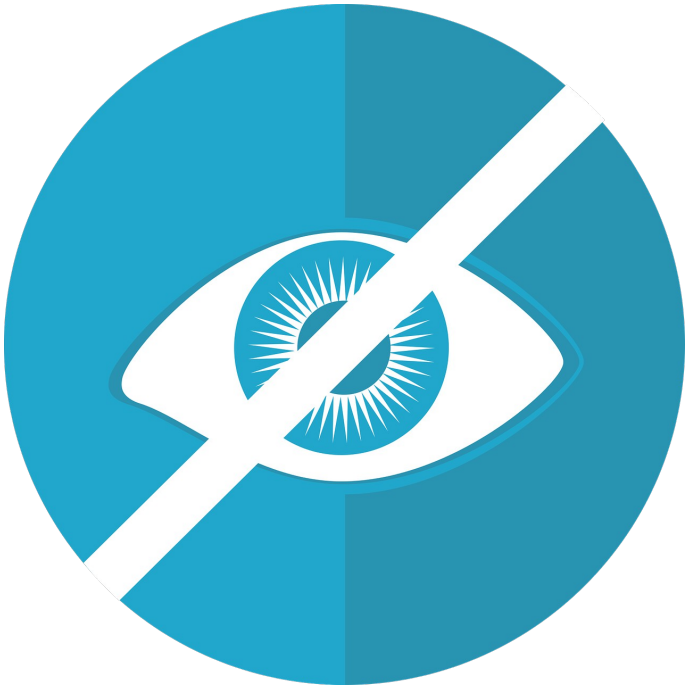
Union-Based SQL Injection



Union-Based SQL Injection uses the SQL UNION operator with a SELECT statement in order to output additional database information from the web app.

Blind SQL Injection

Blind SQL Injection is SQL injection where the error messages resulting from improper SQL queries has been disabled, but the requests are going through nonetheless.



Blind SQL Injection: Auth Bypass

Authorization Bypass is a type of Blind SQL Injection where insecurely written code allows the login of a user to a web app without the use of a proper username and / or password.

Blind SQL Injection: Auth Bypass

```
' or 1=1 --
```

The most basic auth bypass SQL command is illustrated by the SQL command show above. Note that there is a space present in the command after the double dashes (--).

TryHackMe

We'll be continue learning SQL with TryHackMe at the following URL:

https://tryhackme.com/r/room/sqlinjectionlm

Blind SQL Injection: Boolean Based

Boolean Based SQL Injection is a type of Blind SQL Injection where the only output that is returned after a SQL statement is whether the query returned true or false.

Blind SQL Injection: Boolean Based

```
like '%'
```

The key to using Boolean Based SQL Injection is the use of wildcard search queries to incrementally determine valid database contents.

Boolean Based SQL Injection

Boolean-based SQL injection is a SQL injection attack where certain statements are confirmed true or false by the SQL system,

Boolean Based SQL Injection

and in doing so, allows us to enumerate the database's structure and contents.

Boolean Based SQL Injection

Boolean SQL injection is heavily dependant on the LIKE operator and % wildcard character.

Blind SQL Injection: Time Based

Time Based SQL Injection is a type of Blind SQL Injection where we give the database software an instruction to return a delayed response, and if so, we know that the attack was successful.

Blind SQL Injection: Time Based

```
UNION SELECT SLEEP(5)
```

The SQL feature that we use in time-based SQL injection is the **sleep** function, where the number of seconds the response is delayed by is indicated in the parentheses.

Time Based SQL Injection

```
' UNION SELECT SLEEP(5),2 where database() like '%';--
```

Time based SQL injection is similar to boolean based injection in most ways

Time Based SQL Injection

```
' UNION SELECT SLEEP(5),2 where database() like '%';--
```

Except that there is no visual feedback to indicate whether or not an injection string has returned True or False

Time Based SQL Injection

Rather, time based SQL injection makes use of the SQL sleep method to delay the return of successful injection strings

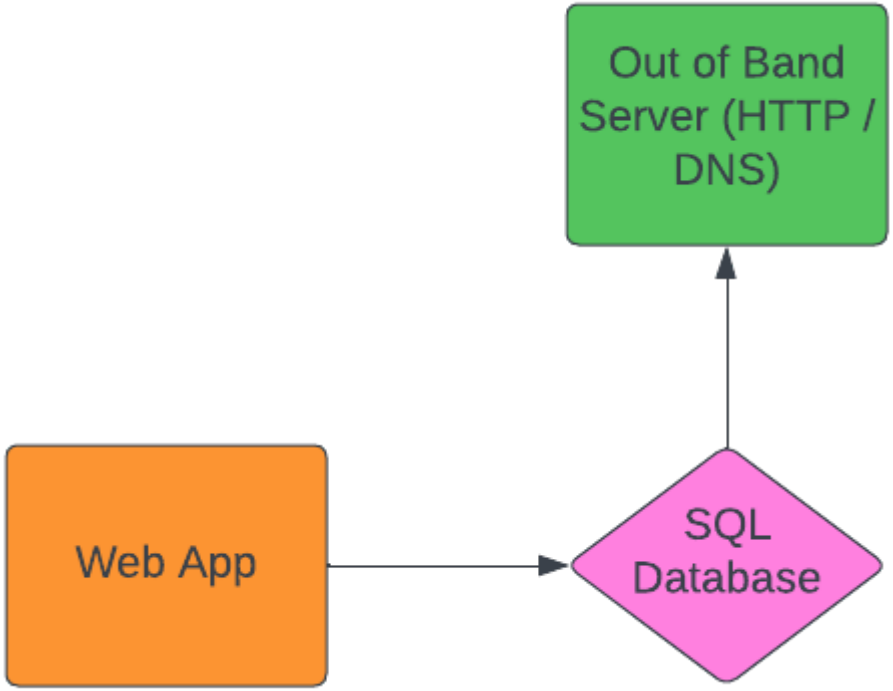
Out-Of-Band (OOB) SQL Injection

Out-Of-Band (OOB) SQL Injection is a type of SQL injection where the output of any SQL queries can be passed to services outside of the web app's network.

Out-Of-Band (OOB) SQL Injection

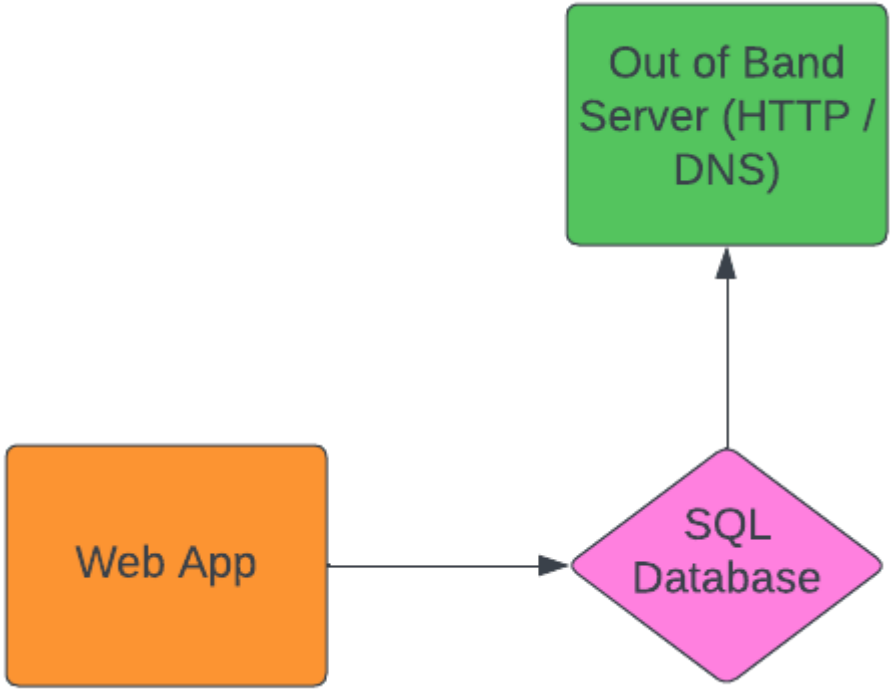
One example would be where SQL injection is able to trigger a DNS query lookup to an attacker-controlled server. SQL query output would then be included in logs for the OOB service used.

Out-Of-Band (OOB) SQL Injection



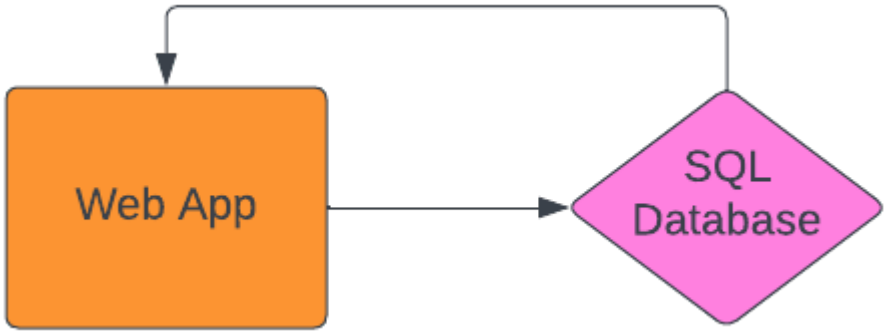
Out of band SQL injection is uncommon because it requires the SQL software to be able to send SQL data “out of band”,

Out-Of-Band (OOB) SQL Injection



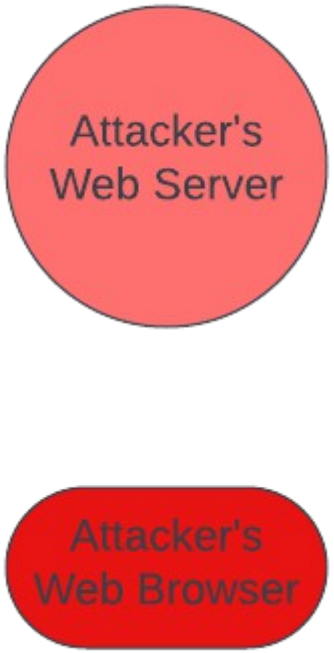
that is, outside of the web app's domain.

Out-Of-Band (OOB) SQL Injection



As opposed to in-band or blind SQL injection, where results of database queries can be “seen” in the web app.

Out-Of-Band (OOB) SQL Injection

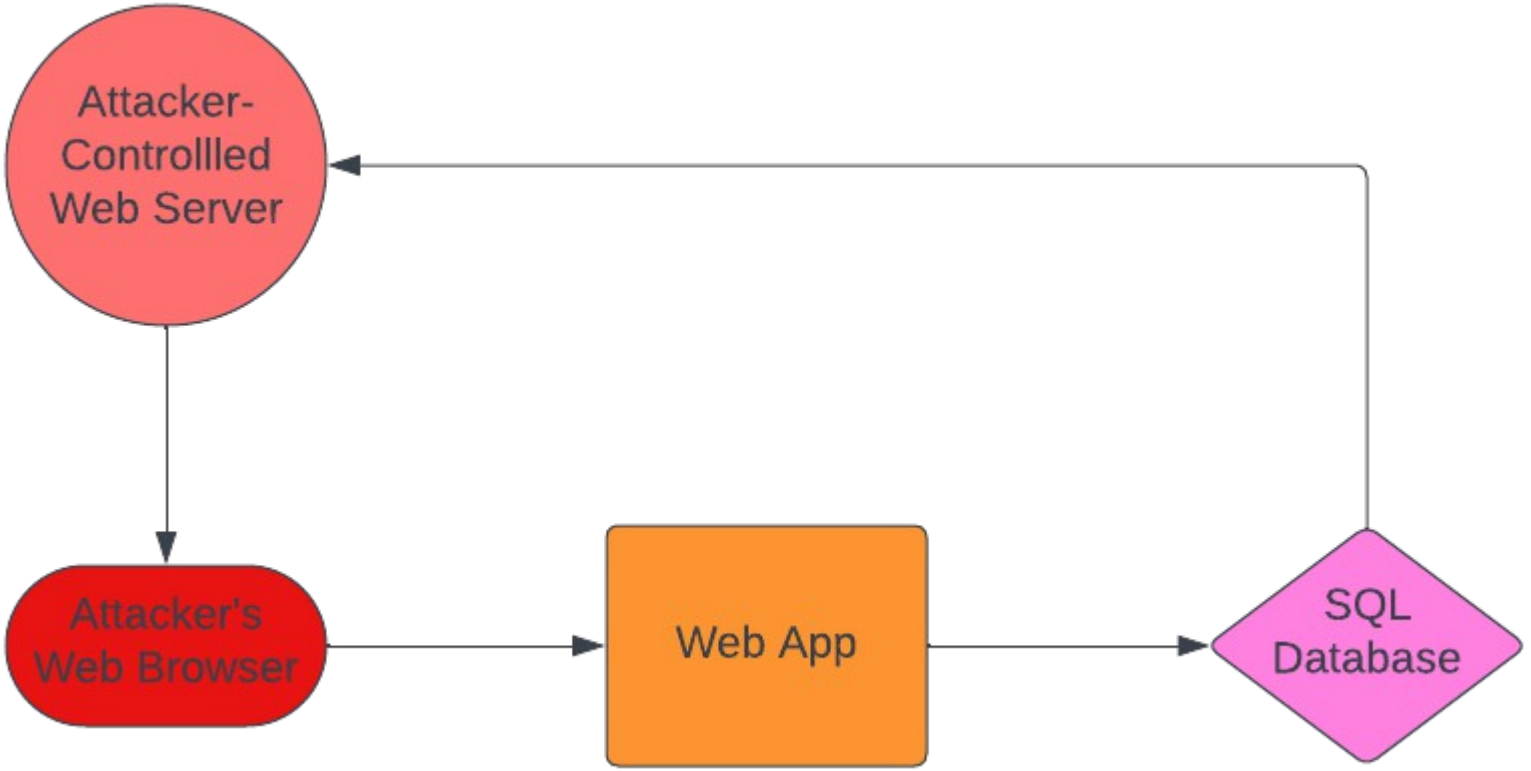


OOB attacks require two different communication channels, one to launch the attack, and another to gather the results.

Out-Of-Band (OOB) SQL Injection

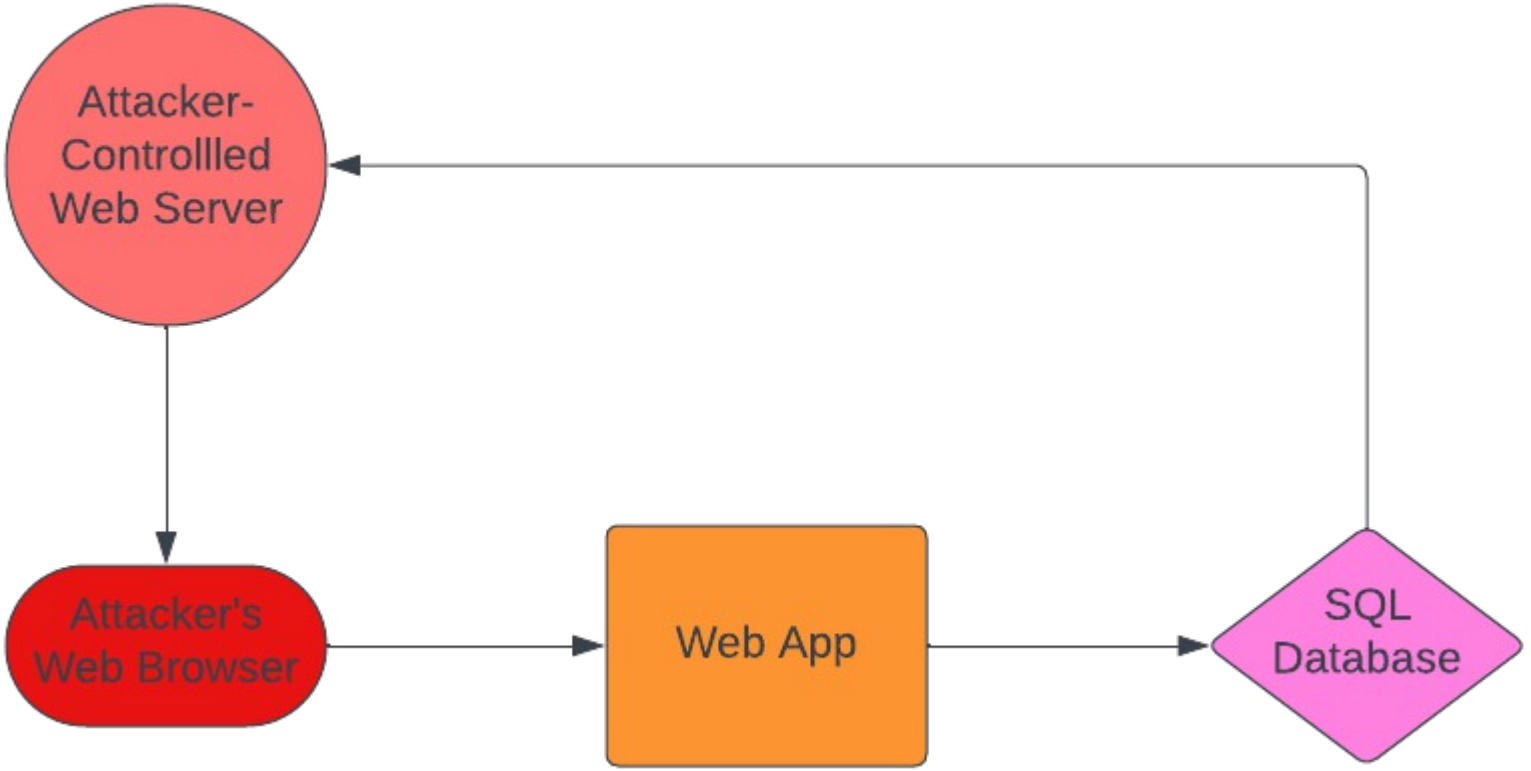
In a “typical” OOB attack, the attacker injects SQL queries to the SQL database via the web app.

Out-Of-Band (OOB) SQL Injection



The database then sends the output of the SQL injection via HTTP / DNS traffic to the attacker's web server.

Out-Of-Band (OOB) SQL Injection



Which receives and logs the results, and can be retrieved by the attacker at their leisure.

Remediation

This section goes over some of the preventative measures developers can take to better secure their web applications against SQL attacks.

Remediation

There are three measures of remediation recommended here.

Prepared Statements with Parameterized Queries

```php
// Prepare the SQL statement with placeholders
$sql = "SELECT * FROM users WHERE username = :username AND password
$stmt = $conn->prepare($sql);

// Bind parameters to placeholders
$stmt->bindParam(':username', $user_input_username);
$stmt->bindParam(':password', $user_input_password);

// Execute the prepared statement
$stmt->execute();
```

The first measure is **prepared statements** with **parameterized queries**.

Prepared Statements with Parameterized Queries

```php
// Prepare the SQL statement with placeholders
$sql = "SELECT * FROM users WHERE username = :username AND password
$stmt = $conn->prepare($sql);

// Bind parameters to placeholders
$stmt->bindParam(':username', $user_input_username);
$stmt->bindParam(':password', $user_input_password);

// Execute the prepared statement
$stmt->execute();
```

Prepared statements are SQL code written in such a way that the user input is compartmentalized away from the SQL query.

Prepared Statements with Parameterized Queries

```php
// Prepare the SQL statement with placeholders
$sql = "SELECT * FROM users WHERE username = :username AND password
$stmt = $conn->prepare($sql);

// Bind parameters to placeholders
$stmt->bindParam(':username', $user_input_username);
$stmt->bindParam(':password', $user_input_password);

// Execute the prepared statement
$stmt->execute();
```

In addition, the **queries** are **parameterized**, which further serves to isolate user input from SQL statements.

Input Validation

```
# Function to sanitize user input to prevent SQL injection
def sanitize_input(input_str):
    # List of special characters commonly used in SQL injection
    sql_injection_chars = [";", "'", "--", "/*", "*/", "xp_cmdshell",

    # Check if any of the special characters are present in the input
    for char in sql_injection_chars:
        if char in input_str:
            return False
    return True
```

The second suggested remediation method is **input validation**, which filters or modifies user input that includes special characters,

Input Validation

```
# Function to sanitize user input to prevent SQL injection
def sanitize_input(input_str):
    # List of special characters commonly used in SQL injection
    sql_injection_chars = [";", "'", "--", "/*", "*/", "xp_cmdshell",

    # Check if any of the special characters are present in the input
    for char in sql_injection_chars:
        if char in input_str:
            return False
    return True
```

Special characters in user input are what enable SQL injection attacks, and input validation helps in securing web apps.

Escaping User Input

```
# Function to escape special characters in user input
def escape_special_characters(input_str):
    # Define a dictionary of special characters and their escape sequences
    special_characters = {
        "&": "&amp;",
        "<": "&lt;",
        ">": "&gt;",
        '"': "&quot;",
        "'": "&#x27;",
        "/": "&#x2F;"
    }
```

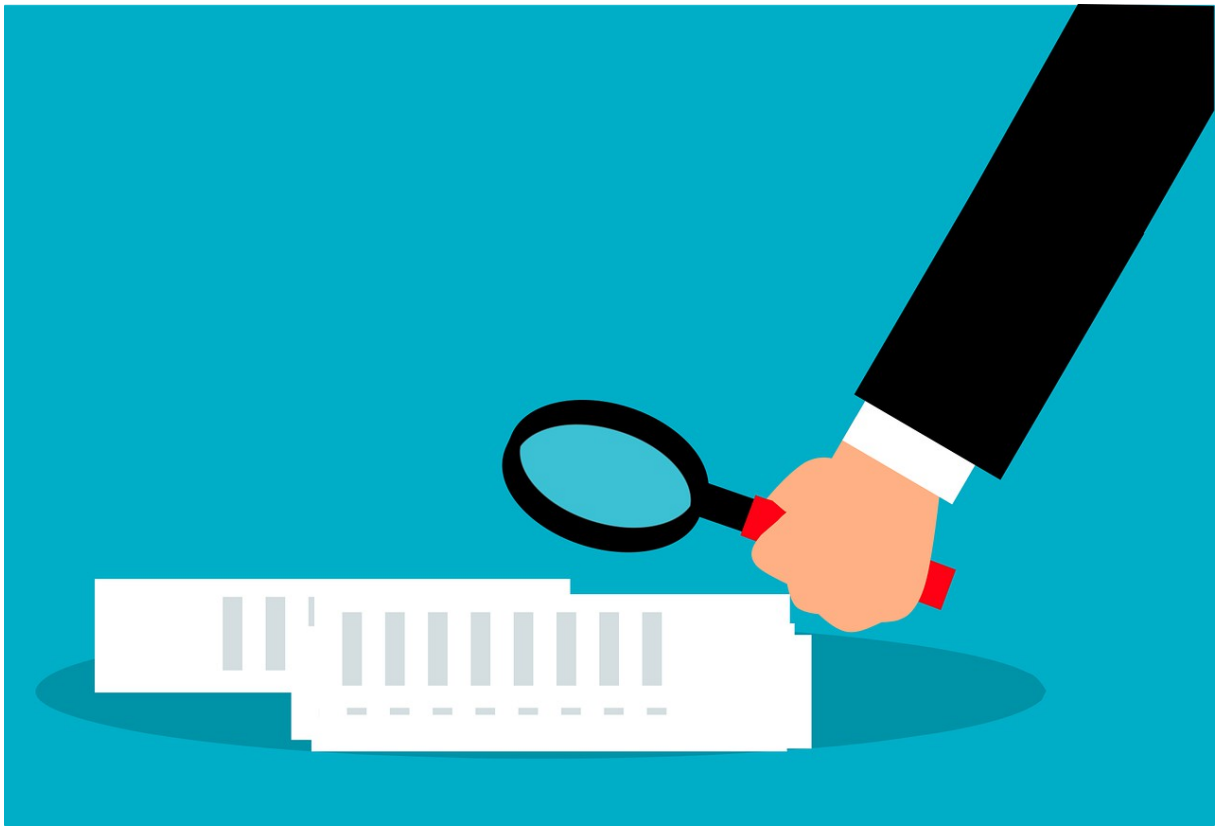
The final suggested remediation method is **escaping user input**, which also targets special characters in user input.

Escaping User Input

```
# Function to escape special characters in user input
def escape_special_characters(input_str):
    # Define a dictionary of special characters and their escape sequences
    special_characters = {
        "&": "&amp;",
        "<": "&lt;",
        ">": "&gt;",
        '"': "&quot;",
        "'": "&#x27;",
        "/": "&#x2F;"
    }
```

But escaped user input modifies special characters in such a way that SQL software will not interpret these characters as code.

Summary



Let's review the SQL concepts we learned in today's workshop:

Blind SQL Injection: Boolean Based

Boolean Based SQL Injection is a type of Blind SQL Injection where the only output that is returned after a SQL statement is whether the query returned true or false.

Blind SQL Injection: Time Based

Time Based SQL Injection is a type of Blind SQL Injection where we give the database software an instruction to return a delayed response, and if so, we know that the attack was successful.

Out-Of-Band (OOB) SQL Injection

Out-Of-Band (OOB) SQL Injection is a type of SQL injection where the output of any SQL queries can be passed to services outside of the web app's network.

Remediation

Lastly, we learned about how to remediate web apps against SQL injection vulnerability.

Remediation

With practices such as **prepared statements** with **parameterized queries**, **input validation**, and **escaping user input**.

What's Next?

In the next HackerFrogs AfterSchool web app hacking workshop, we'll be learning how to use Burpsuite, the industry-standard software for web app security testing.

Extra Credit

Looking for more study material on this workshop's topics?

See this video's description for links to supplemental documents and exercises!

Until Next Time, HackerFrogs!