Part 4 Vulnerabilities (Bad)

Penguin Union

Step1

On the challenge server, I used `a' OR address WHERE 1=1`

Step2

I used the SQL query: `a' UNION SELECT address, null FROM registrations --"` and retrieved the flag:

Flag Found

```
UWA{tH4t5_sOMe_b3ZOs_lVl_vN1oN_bUsTin}
```

Part 4 Vulnerabilities (Average)

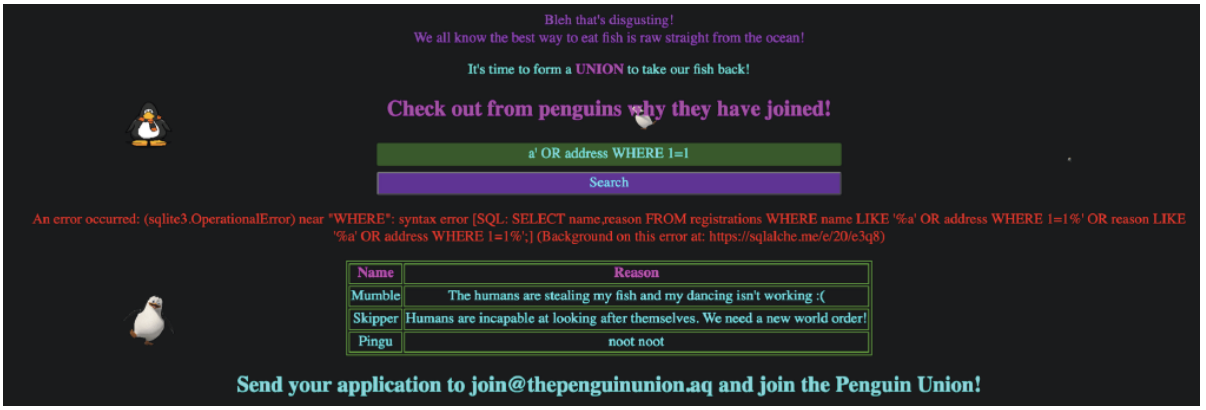
Penguin Union

Step 1

Intentionally typing the wrong SQL statement in an attempt to obtain the correct database name.

```
a' OR address where 1=1
```

Result:



This attempt inadvertently revealed the table name (`registrations`) and column names (`name` and `reason`).

Step 2

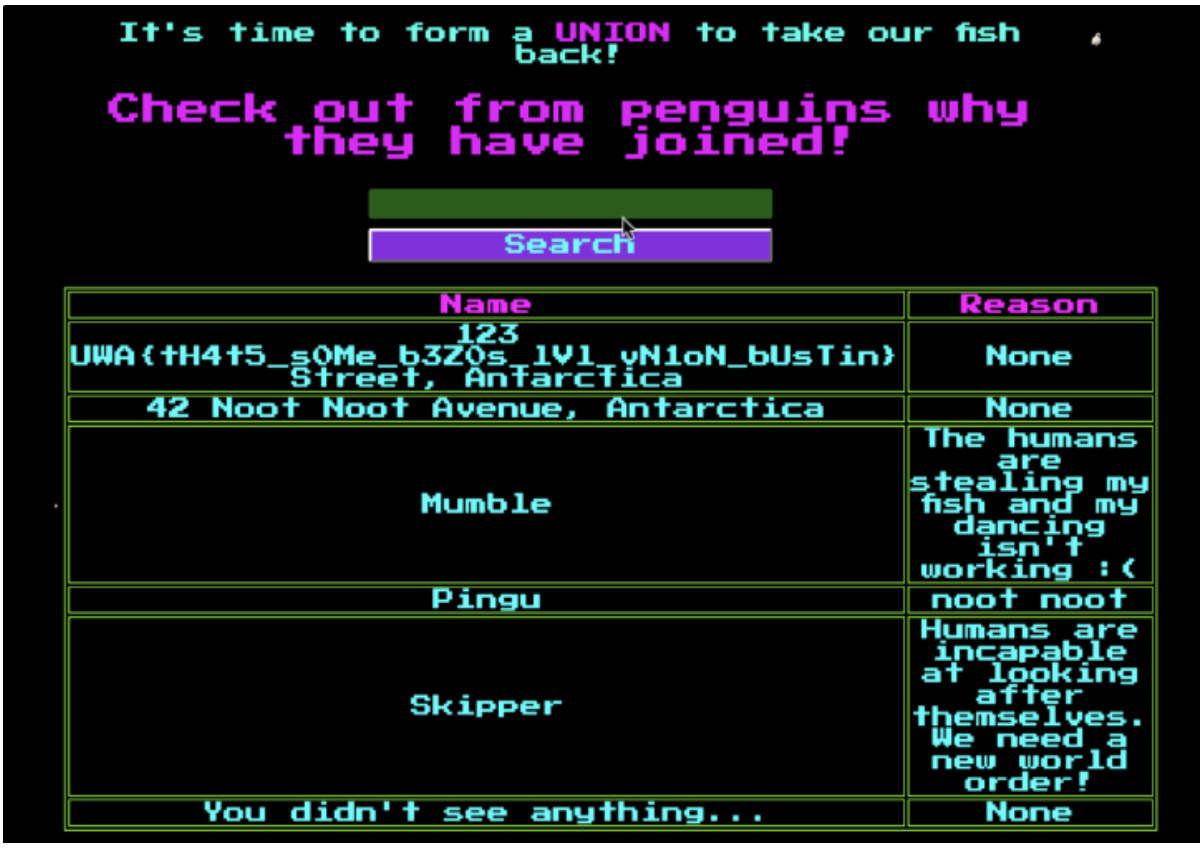
I noticed a hint suggesting the use of a `UNION` operation: `It's time to form a UNION to take our fish back!`. A `UNION` operation in SQL allows combining the results of two or more `SELECT` statements into a single result set.

I crafted an SQL query as follows:

```
a' UNION SELECT address, null FROM registrations --"
```

Where `a'` marks the end of the original query and serves as the starting point for injection. `UNION SELECT address, null FROM registrations` retrieves the `address` column from the `registrations` table. The inclusion of `null` in the second column ensures consistency in the number of columns returned by both `SELECT` statements. The trailing `--` comments out any remaining portion of the original query to prevent syntax errors.

Result:



Flag Found:

```
UWA{tH4t5_sOMe_b3ZOs_lVl_vN1oN_bUsTin}
```

Part 4 Vulnerabilities (Good)

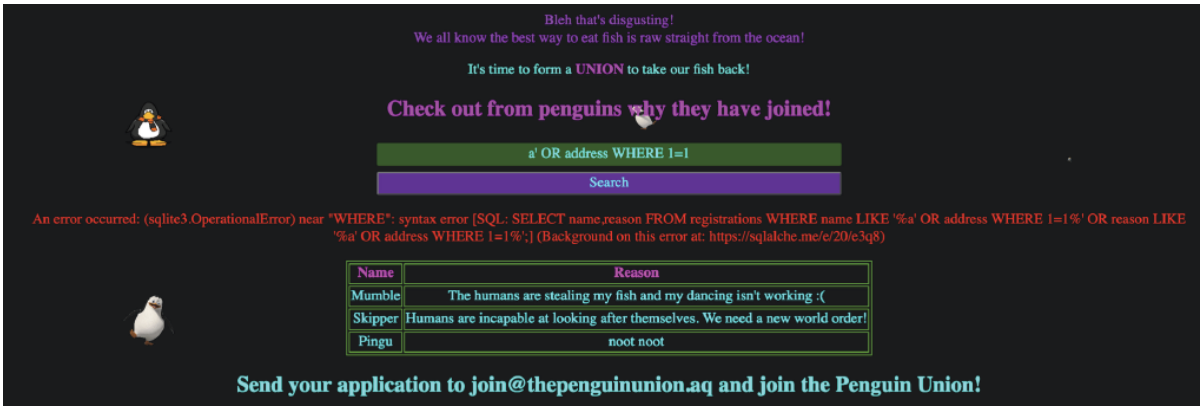
Penguin Union

Step 1

In an attempt to exploit the SQL vulnerability, I formulated the following input:

```
a' OR address WHERE 1=1
```

Result:



Step 2

The attempted SQL query is as follows:

```
SELECT name, reason FROM registrations WHERE name LIKE '%a' OR address WHERE
1=1%' OR reason LIKE 'a' OR address WHERE 1=1%';
```

The query is explained as below:

- **SELECT:** It specifies the columns `name` and `reason` to be retrieved from the `registrations` table.
- **FROM:** This clause identifies the table being queried, which is `registrations`.
- **WHERE:** Here are the issues encountered:
  - The condition `LIKE '%a'` suggests a search for records where the `name` column ends with the letter 'a', which seems valid.
  - The segment `OR address WHERE 1=1%'` appears to be a mix-up, possibly attempting to inject a condition using `WHERE 1=1%`, which is syntactically incorrect.
  - The condition `OR reason LIKE 'a'` indicates a search for records where the `reason` column exactly matches the letter 'a', which seems valid.
  - The segment `OR address WHERE 1=1%` also seems to be a mix-up with conditions, similar to the preceding segment.

Step 3

In Step 2, I gained insights into the table schema and noticed a hint suggesting the use of a `UNION` operation: `It's time to form a UNION to take our fish back!`. `UNION` enables merging the results of multiple `SELECT` queries and thus expanding the scope of the original query to include additional data fields.

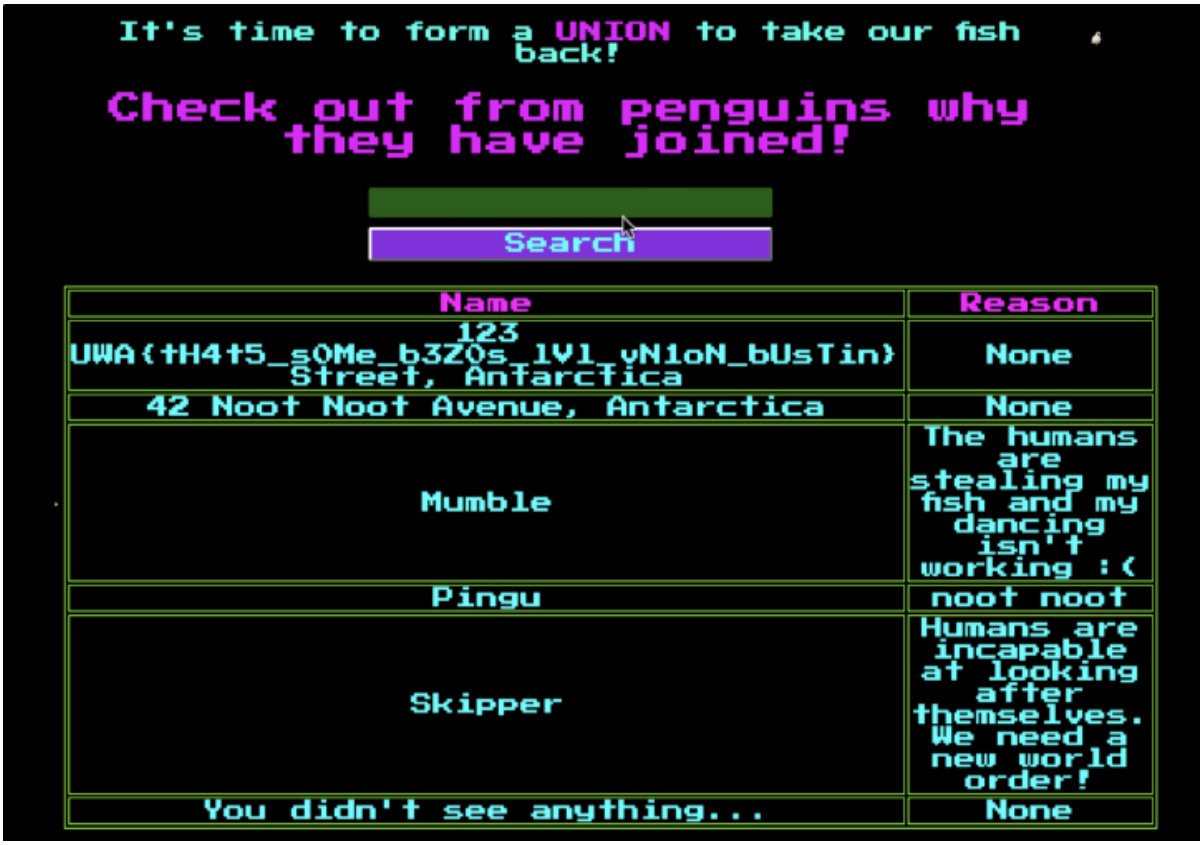
Crafted Input:

```
a' UNION SELECT address, null FROM registrations --"
```

Explanation:

- `a'`: Marks the end of the original query and serves as the starting point for injection.
- `UNION SELECT address, null FROM registrations`: Introduces a new `SELECT` statement that retrieves the `address` column from the `registrations` table. The inclusion of `null` in the second column ensures consistency in the number of columns returned by both `SELECT` statements.
- The trailing `--` comments out any remaining portion of the original query to prevent syntax errors.

Result:



Flag Found:

```
UWA{tH4t5_sOMe_b3ZOs_lVl_vN1oN_bUsTin}
```

Part 4 Vulnerabilities (Excellent)

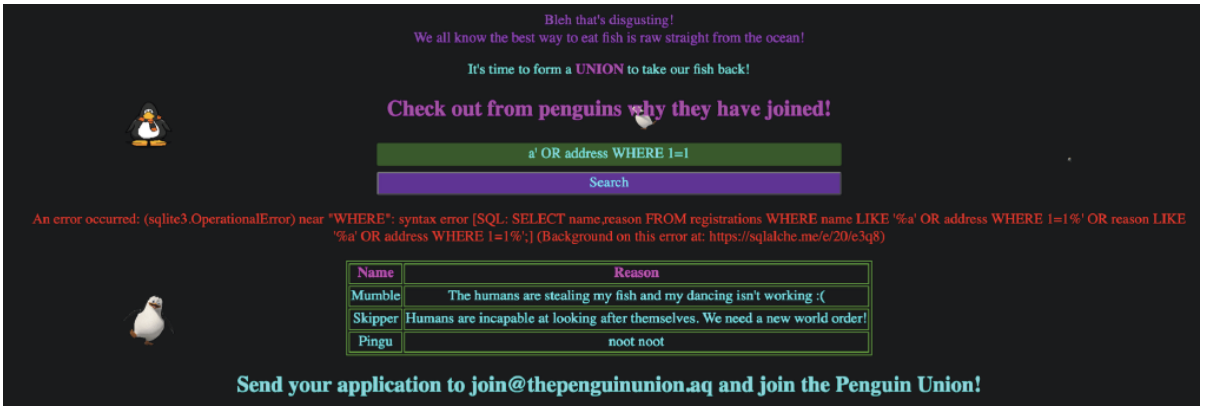
Penguin Union

Step 1: Attempting an SQL Injection

In an attempt to exploit the SQL vulnerability, I formulated the following input:

```
a' OR address WHERE 1=1
```

Result:



This attempt inadvertently revealed the table name (`registrations`) and column names (`name` and `reason`). Such revelations provide insights into the database structure, which can be exploited.

Step 2: Analyzing the attempted SQL Query

The attempted SQL query is as follows:

```
SELECT name, reason FROM registrations WHERE name LIKE '%a' OR address WHERE
1=1%' OR reason LIKE 'a' OR address WHERE 1=1%';
```

Let's dissect this query:

- **SELECT:** It specifies the columns `name` and `reason` to be retrieved from the `registrations` table.
- **FROM:** This clause identifies the table being queried, which is `registrations`.
- **WHERE:** Here are the issues encountered:
  - The condition `LIKE '%a'` suggests a search for records where the `name` column ends with the letter 'a', which seems valid.
  - The segment `OR address WHERE 1=1%'` appears to be a mix-up, possibly attempting to inject a condition using `WHERE 1=1%`, which is syntactically incorrect.
  - The condition `OR reason LIKE 'a'` indicates a search for records where the `reason` column exactly matches the letter 'a', which seems valid.
  - The segment `OR address WHERE 1=1%` also seems to be a mix-up with conditions, similar to the preceding segment.

Step 3: Crafting Input

In Step 2, I gained insights into the table schema and noticed a hint suggesting the use of a `UNION` operation: `It's time to form a UNION to take our fish back!`. A `UNION` operation in SQL allows combining the results of two or more `SELECT` statements into a single result set.

To expose the addresses of registered penguins, I opted to employ the `UNION` operation. Employing an SQL injection featuring `UNION` is to extract the addresses stored within the `registrations` table. Here's the rationale:

1. **Combining Queries**: `UNION` enables merging the results of multiple `SELECT` queries. In this case, I wanted to retrieve the addresses from the `registrations` table, and using `UNION` allowed me to achieve this by combining the original query with a new `SELECT` statement targeting the `address` column.
2. **Enhancing Data Retrieval**: Incorporating `UNION` could include additional data fields. This flexibility is particularly useful when attempting to retrieve specific information, such as addresses, that may not be directly accessible through the original query alone.

In my SQL injection attempt, I leveraged the `'` symbol to inject a custom query alongside the `UNION` operation. Let's break down the components of the crafted input:

Crafted Input:

```
a' UNION SELECT address, null FROM registrations --"
```

Explanation:

- `a'`: Marks the end of the original query and serves as the starting point for injection.
- `UNION SELECT address, null FROM registrations`: Introduces a new `SELECT` statement that retrieves the `address` column from the `registrations` table. The inclusion of `null` in the second column ensures consistency in the number of columns returned by both `SELECT` statements.
- The trailing `--` comments out any remaining portion of the original query to prevent syntax errors.

This injection combines the original query with the additional `SELECT` statement, leveraging the `UNION` operation to extract the desired addresses from the `registrations` table.

My expected SQL statement is as follows:

```
SELECT name, reason FROM registrations WHERE name LIKE '%a' UNION SELECT address,
null FROM registrations --%' OR reason LIKE 'a' OR address WHERE 1=1%';
```

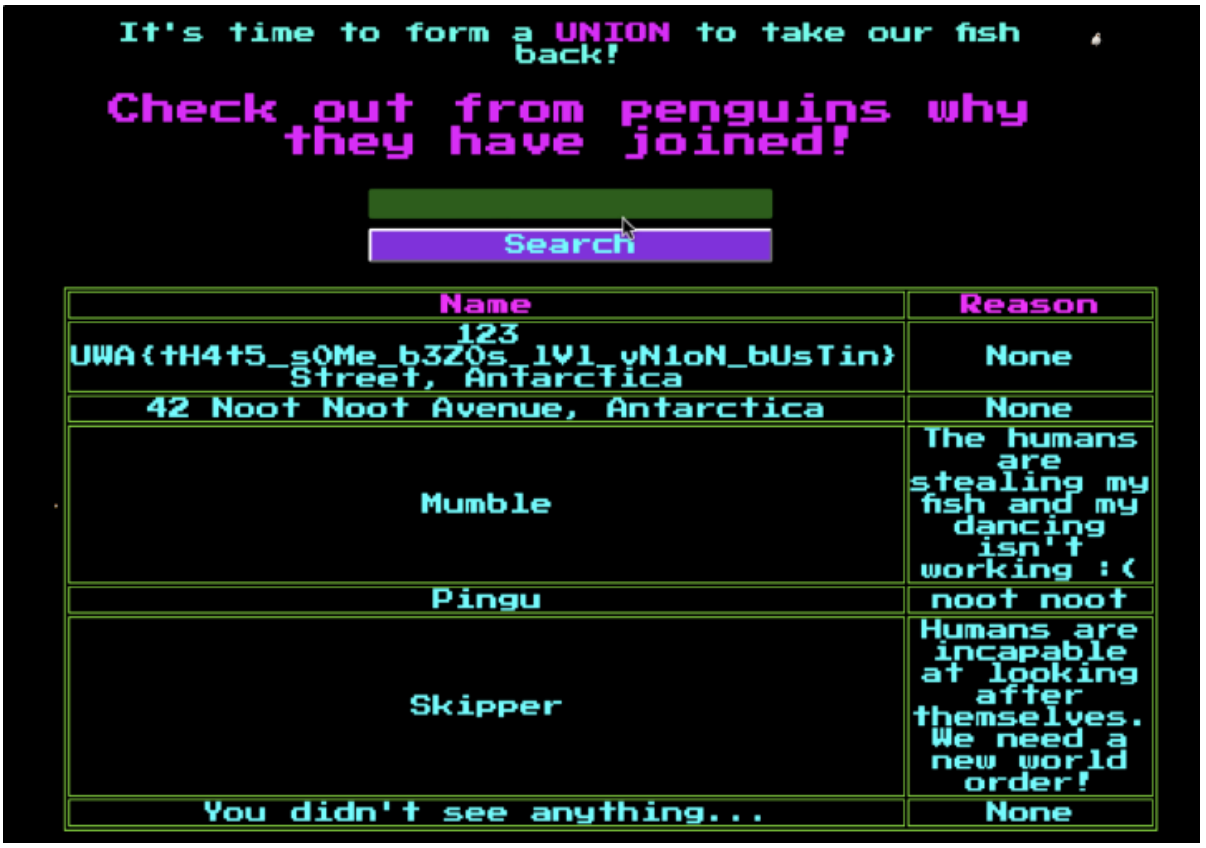
Let's dissect this statement:

- The original query concludes with `'`, indicating the termination of the initial SQL command and the commencement of the injection.
- `UNION SELECT address, null FROM registrations` introduces a new `SELECT` statement that retrieves the `address` column from the `registrations` table. The inclusion of `null` in the second column ensures uniformity in the number of columns returned by both `SELECT` statements.

- The trailing `--` comments out any subsequent portion of the original query to prevent syntax errors.

This SQL statement integrates the original query with the injected `UNION` operation to extract addresses from the `registrations` table.

Result:



Flag Found:

```
UWA{tH4t5_sOMe_b3ZOs_lVl_vN1oN_bUsTin}
```