ALGORITHM

Classic injection bypass:

Algorithm

1. Start
2. User Enters Username and Password
3. Validate User Input (Optional)

   a. If (true)

   Valid, Proceed to Next Step

   b. else (false)

   Invalid, Return Error and Require Correct Input
4. Check for Injection Bypass Attempt admin'OR '1'='1

   a. If (true)

   User Input Detected as an Injection Attempt, Return Access Denied Error

   b. else (false)

   User Input is Valid, Proceed to Next Step
5. Attempt Access

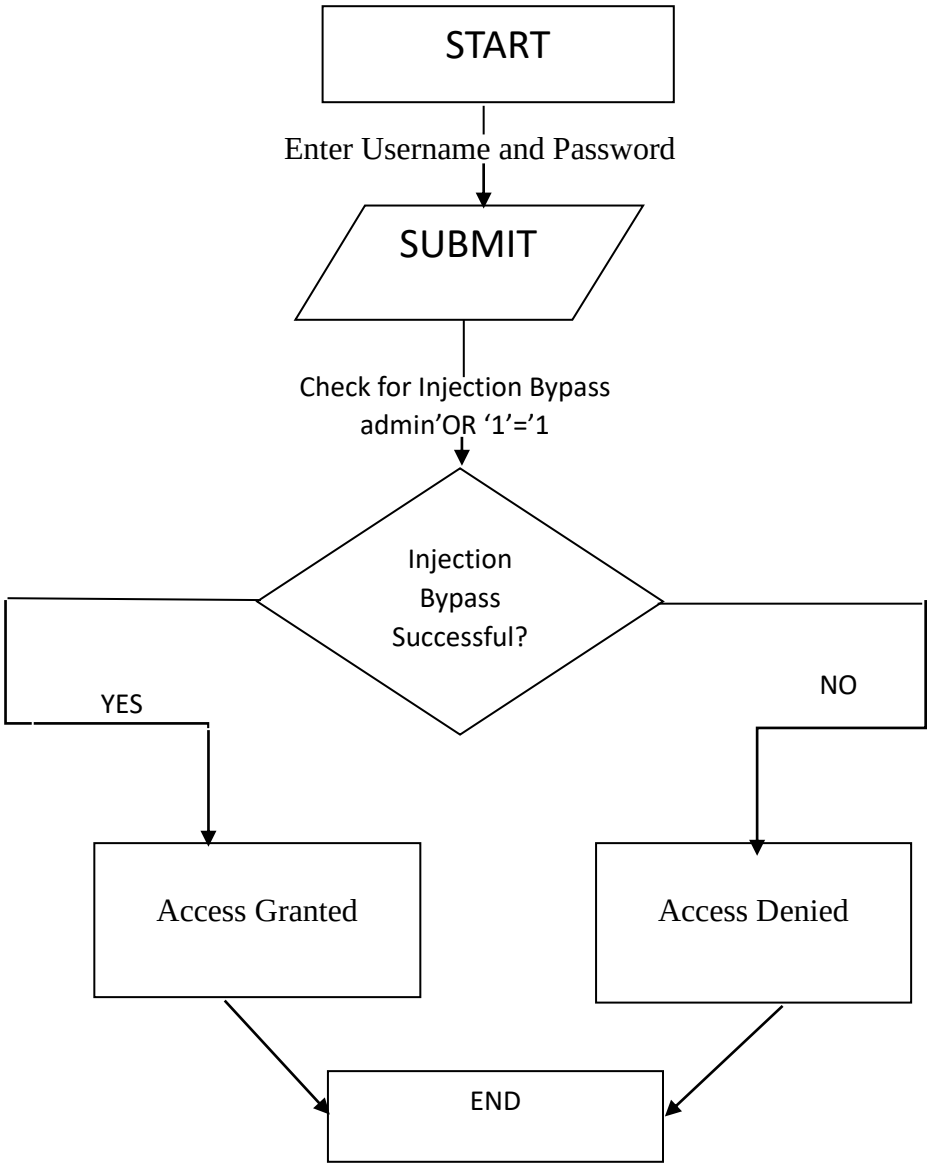
   a. If (true)

   Username and Password Match Database, Return Access Granted

   b. else (false)

   Username and Password Do Not Match, Return Access Denied Error
6. End

Flowchart:


START
Enter Username and Password
SUBMIT
Check for Injection Bypass
admin'OR '1'='1
Injection
Bypass
Successful?
YES
NO
Access Granted
Access Denied
END

Union Based SQLi

Algorithm

1. Is the website vulnerable to SQL injection?

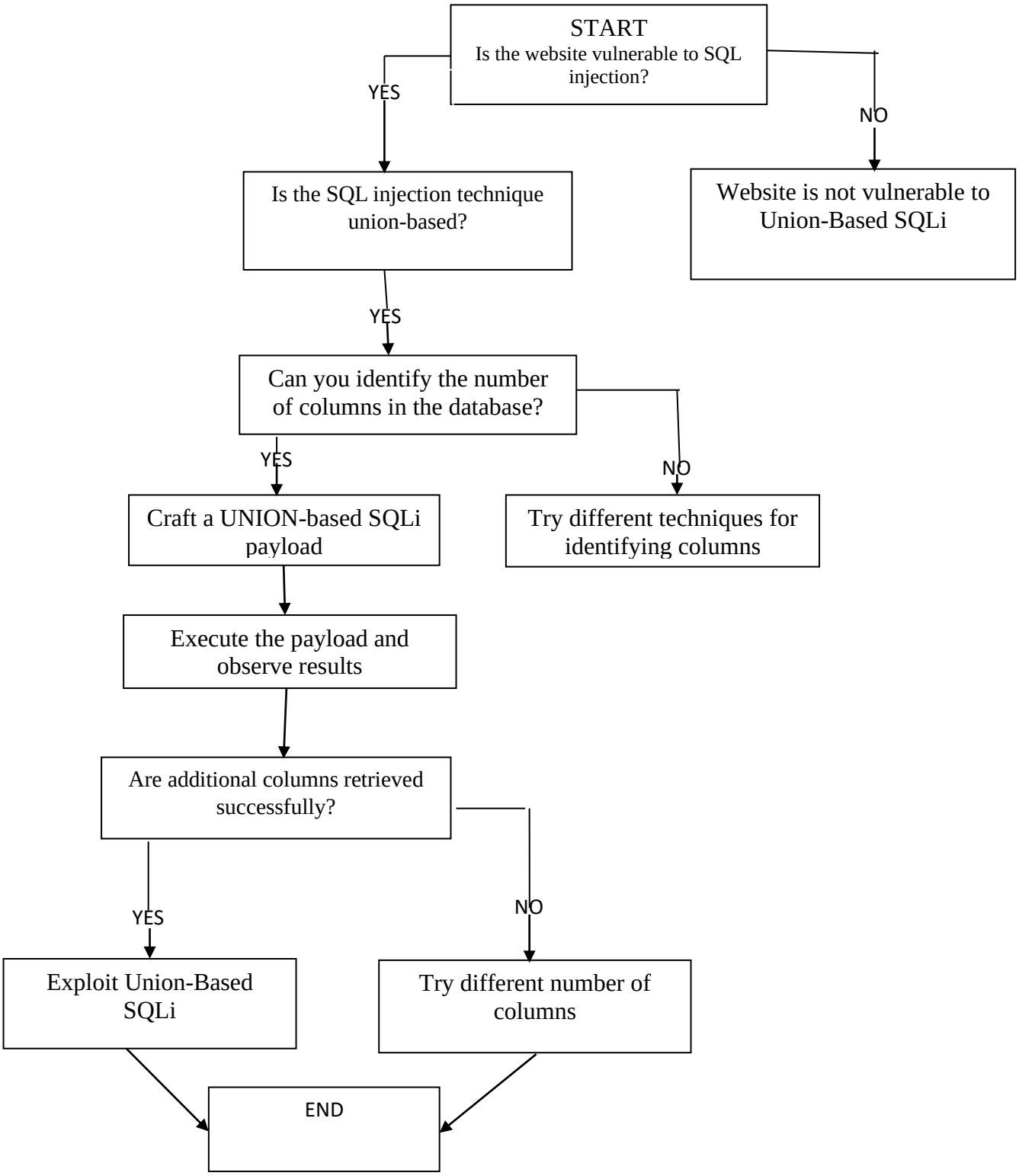
2. If the answer is no, try different techniques for identifying columns.

3. If no technique is successful, try different numbers of columns.

4. If none of the above steps result in successful exploitation, the website

```
function performWebsiteExploitation() {
 var websiteIsVulnerableToSQLInjection = ask("Is the website vulnerable to SQL injection?");
 if (!websiteIsVulnerableToSQLInjection) {
   var columnsIdentifiedSuccessfully = tryDifferentTechniquesForIdentifyingColumns();
   if (!columnsIdentifiedSuccessfully) {
     var columnsNumberIdentifiedSuccessfully = tryDifferentNumbersOfColumns();
     if (!columnsNumberIdentifiedSuccessfully) {
       // None of the above steps resulted in successful exploitation.
       console.log("Website could not be exploited."); }
else {
       // Columns number were identified successfully.
       console.log("Website exploited successfully.");}
   } else {
     // Columns were identified successfully.
     console.log("Website exploited successfully."); }
 } else {
   // The website is vulnerable to SQL injection.
   console.log("Website exploited successfully.");
 }}
```

Flowchart:


START
Is the website vulnerable to SQL injection?
YES
NO
Is the SQL injection technique union-based?
Website is not vulnerable to Union-Based SQLi
YES
Can you identify the number of columns in the database?
YES
NO
Craft a UNION-based SQLi payload
Try different techniques for identifying columns
Execute the payload and observe results
Are additional columns retrieved successfully?
YES
NO
Exploit Union-Based SQLi
Try different number of columns
END

Error Based SQL injection

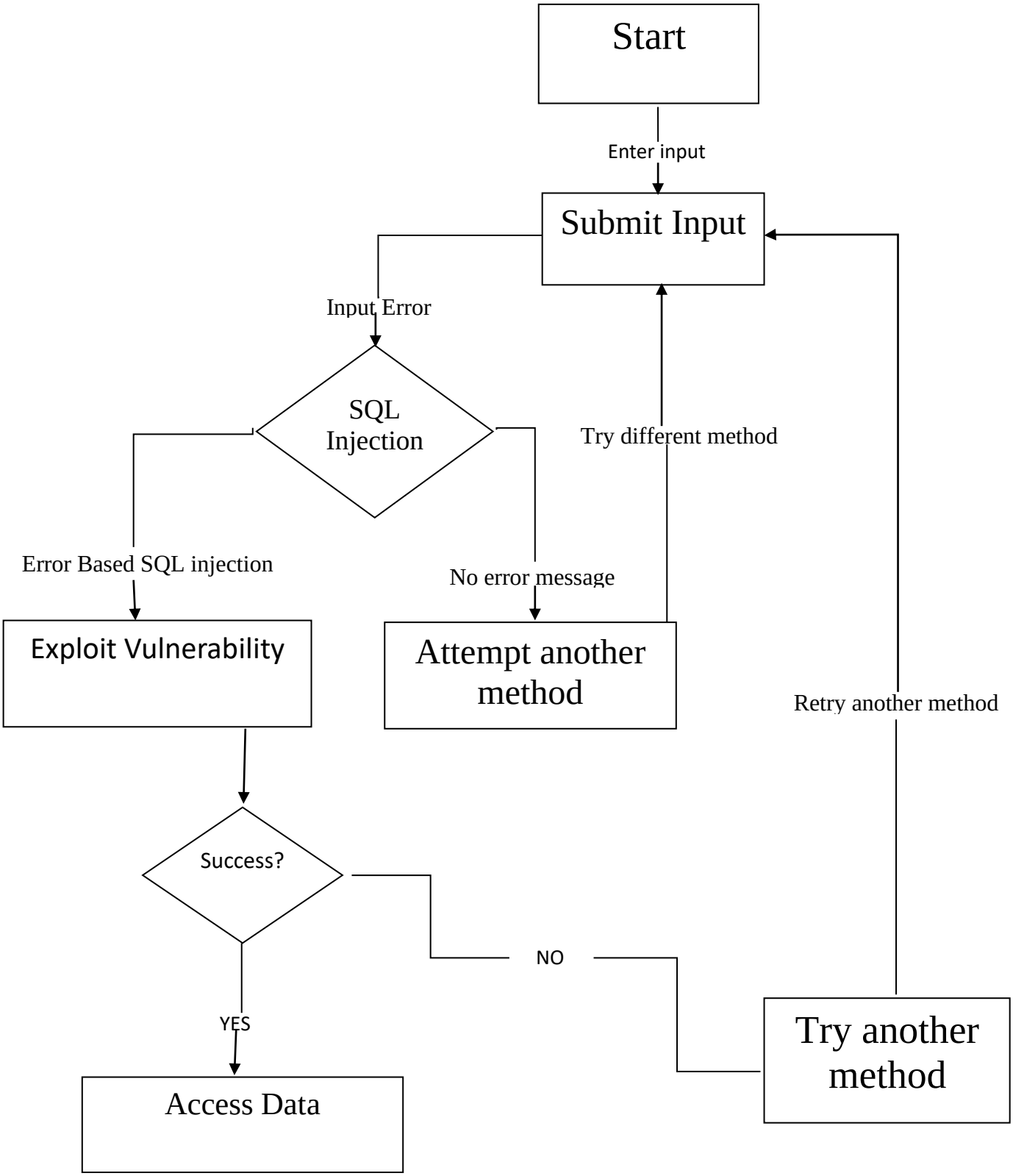
Algorithm

1. Start
2. Enter Input
3. Submit Input
4. Attempt Different Method
   - if failed, go to 5. Retry Another Method
   - if succeeded, go to 7. Exploit Vulnerability (Success)
5. Retry Another Method
   - if failed, go to 6. Try Another Method (T)
   - if succeeded, go to 7. Exploit Vulnerability (Success)
6. Try Another Method (T)

   repeat steps 5-6 until a method that successfully exploits the vulnerability is found

7. Exploit Vulnerability (Success)

Flowchart:


Start
Enter input
Submit Input
Input Error
SQL Injection
Try different method
Error Based SQL injection
No error message
Exploit Vulnerability
Attempt another method
Retry another method
Success?
NO
YES
Try another method
Access Data

Boolean-based SQL injection

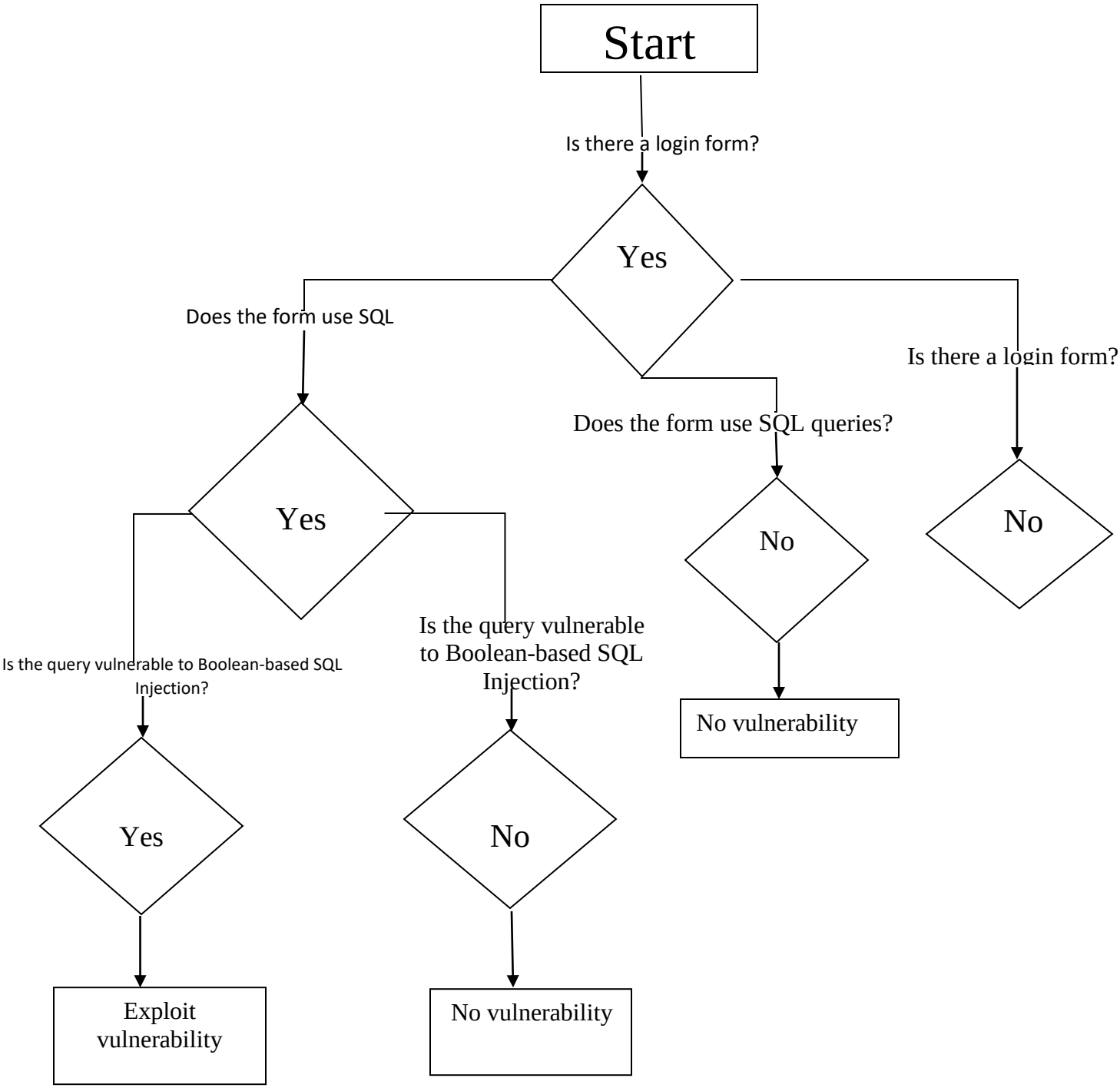
Algorithm

1. start
2. Does the system contain a login form?
   - If Yes, proceed to step 3.
   - If No, terminate the algorithm. The system is not vulnerable to SQL injection attacks.
3. Is the login form designed to use SQL queries?
   - If Yes, proceed to step 4.
   - If No, terminate the algorithm. The system is not vulnerable to SQL injection attacks.
4. Is the login form's SQL query vulnerable to Boolean-based SQL injection attacks?
   - If Yes, proceed to step 5.
   - If No, terminate the algorithm. The system is not vulnerable to SQL injection attacks.
5. If the algorithm identifies a vulnerability, perform the following steps:

    1. Begin a penetration test on the vulnerable login form.
    2. Analyze the login form's SQL query and identify the parameters that can be manipulated.
    3. Develop a Boolean-based SQL injection payload using the identified parameters.
    4. Execute the SQL injection payload on the login form and analyze the response.
    5. If successful, report the vulnerability to the system owner for remediation.
    6. Continue the penetration test to identify and exploit other potential vulnerabilities.

6. If the algorithm does not identify a vulnerability, proceed to step 7.
7. End

Flowchart:


Start
Is there a login form?
Yes
Does the form use SQL
Is there a login form?
Does the form use SQL queries?
Yes
No
No
Is the query vulnerable to Boolean-based SQL Injection?
Is the query vulnerable to Boolean-based SQL Injection?
No vulnerability
Yes
No
Exploit vulnerability
No vulnerability