# NoSQL Injection Vulnerability in Laravel

## Introduction

This document demonstrates a NoSQL injection vulnerability in a Laravel web application, where user input is improperly sanitized, allowing an attacker to bypass authentication. Additionally, it includes an exploitation script and a secure code fix to mitigate the vulnerability.

## Vulnerable Code

The vulnerable code is a login mechanism that allows a NoSQL injection attack on the email field. By failing to properly sanitize user input, the application allows malicious NoSQL operators to manipulate MongoDB queries.

**Vulnerable Code (LoginController.php):**

    public function vulnerableLogin(Request $request)

    {

        $email = $request->input('email');  *// Directly takes user input*

*// Vulnerable query allowing NoSQL injection*

        $emailQuery = json\_decode($email, true);

        $user = User::whereRaw([

            'email' => $emailQuery  *// Unsanitized user input injected here*

        ])->first();

        if ($user) {

            auth()->login($user);

            return redirect('/home')->with('status', 'Welcome ' . $user->name);

        } else {

            return back()->withErrors('Invalid login credentials.');

        }

    }

**Issue**: The email input is not validated or sanitized and is directly injected into the query. This allows an attacker to exploit the NoSQL database (MongoDB) by passing JSON-based query operators (e.g., $gt) into the input fields, bypassing authentication.

{"$gt": ""}

By exploiting this vulnerability, an attacker can bypass the login process without knowing the correct credentials. The attacker can manipulate the email field using NoSQL injection to match any user in the database.

## Security Enhanced Code

To mitigate this vulnerability, proper input validation and sanitization should be applied. Specifically, ensure that the email field is validated as a valid email and that no direct raw inputs are allowed in MongoDB queries.

**Security Enhanced Code (LoginController.php):**

    public function secureLogin(Request $request)

    {

*// Validate input to prevent NoSQL injection*

        $credentials = $request->validate([

            'email' => 'required|email',

            'password' => 'required'

        ]);

*// Use Laravel's attempt method for secure authentication*

        if (auth()->attempt($credentials)) {

            return redirect('/home')->with('status', 'Login successful');

        }

        return back()->withErrors('Invalid login credentials.');

    }

**Fix**: The input is validated using Laravel's built-in validation rules ('email' => 'required|email'). This ensures that the email input is a valid email address and does not contain any malicious query operators.

**Authentication**: Laravel's auth()->attempt() method is used to handle authentication securely, using hashed passwords and avoiding raw query execution.