Національний Технічиний Університет України

«Київський Політехнічний Інститут імені Ігоря Сікорського»

Інститут Прикладного Системного Аналізу

Кафедра Системного Проектування

Безпека інформаційних систем

Звіт про проходження курсу SQL-ін’єкція

Роботу виконав:

Дєрюгін Є.О.

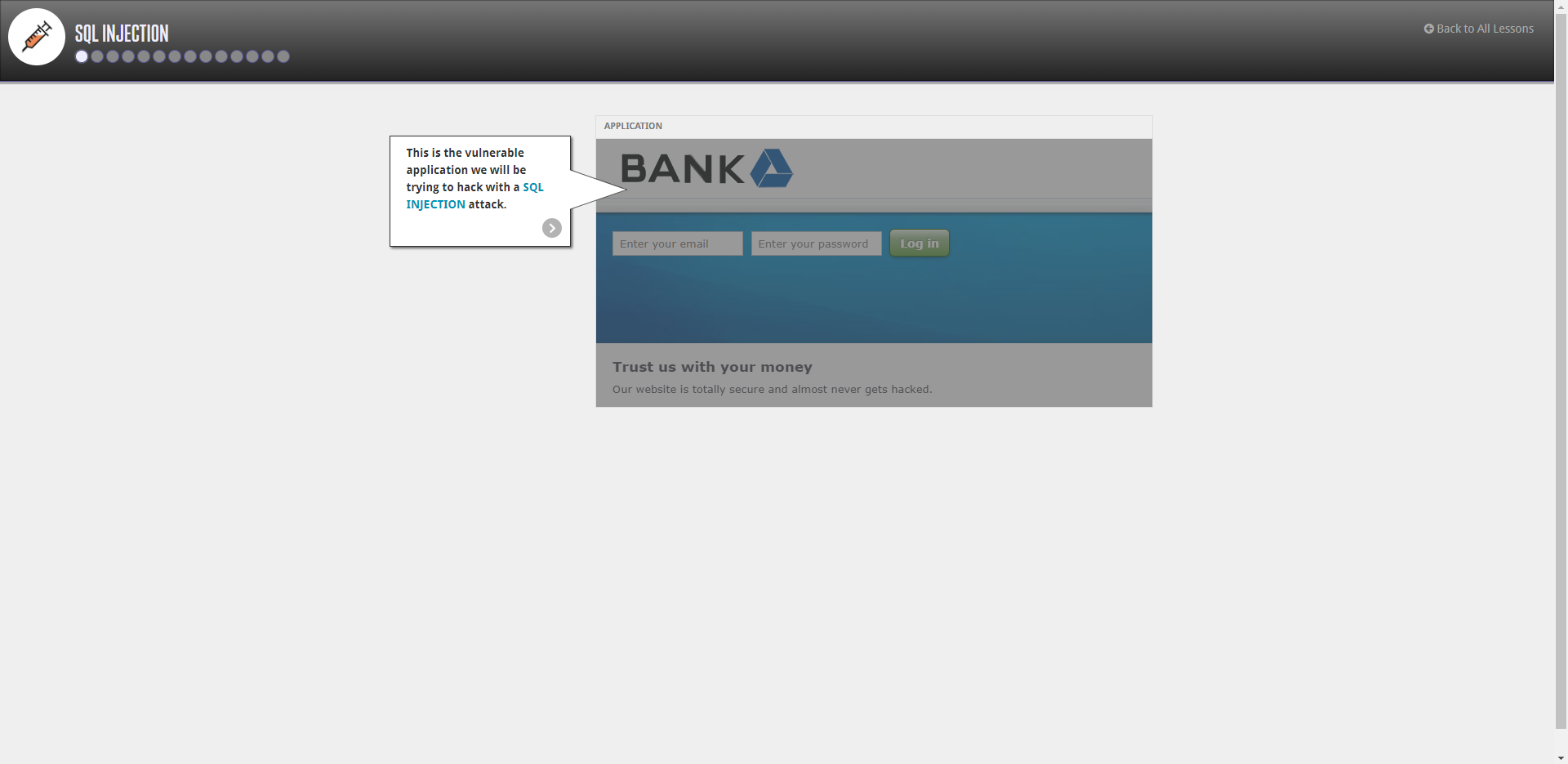
Група: ДА-81

Перевірив: Яременко В. С.

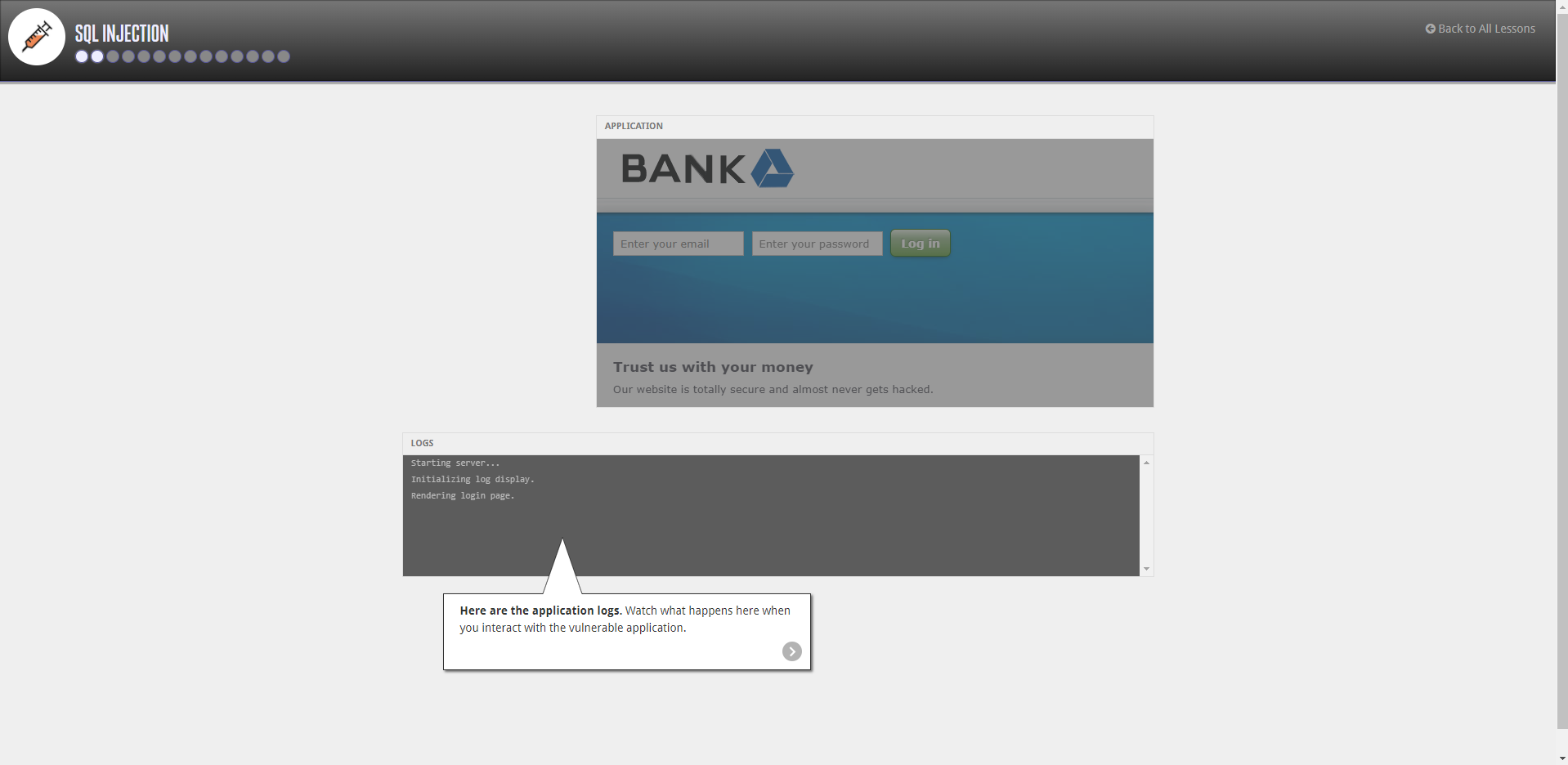
Київ – 2021

**Проходження курсу:**

Це вразливий додаток, який ми спробуємо зламати за допомогою атаки з використанням SQL-ін’єкції.



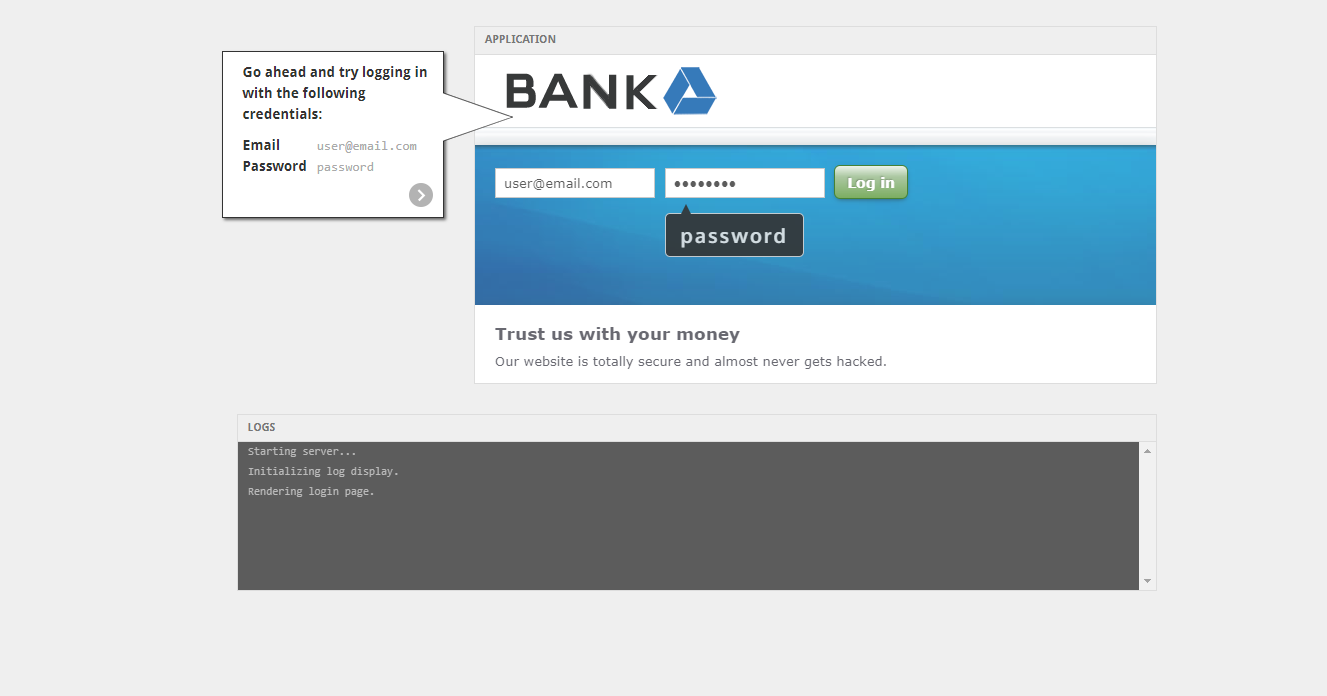
Це логи додатку, де можна побачити що буде відбуватися при користуванні вразливим додатком.



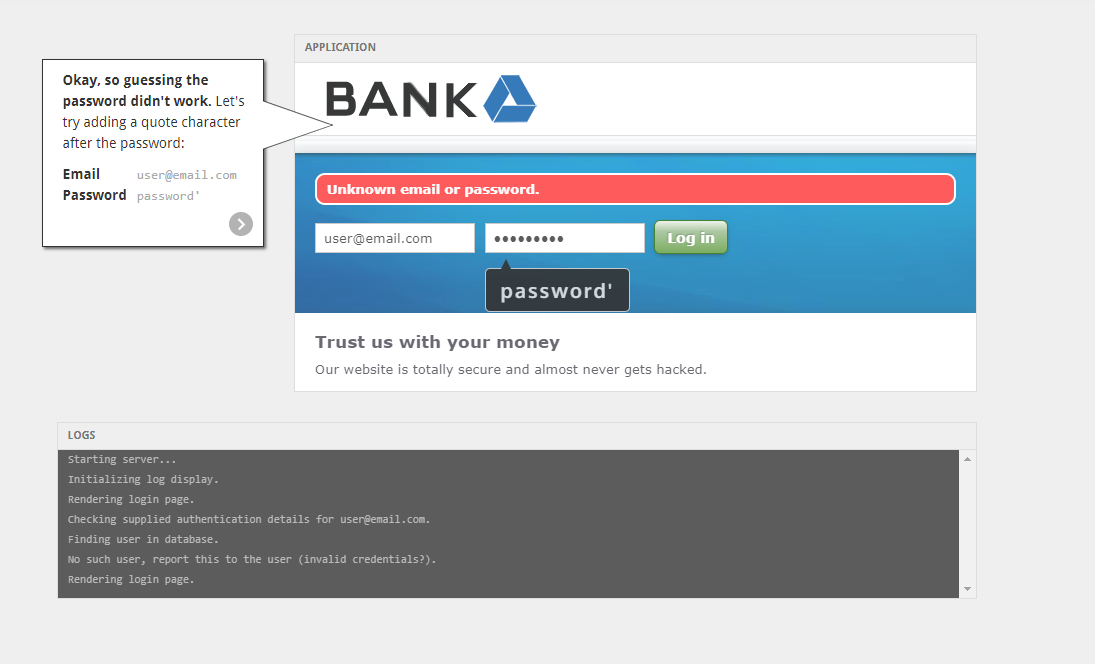
Спробуємо увійти з наступними обліковими даними:

Email: [user@email.com](mailto:user@email.com)

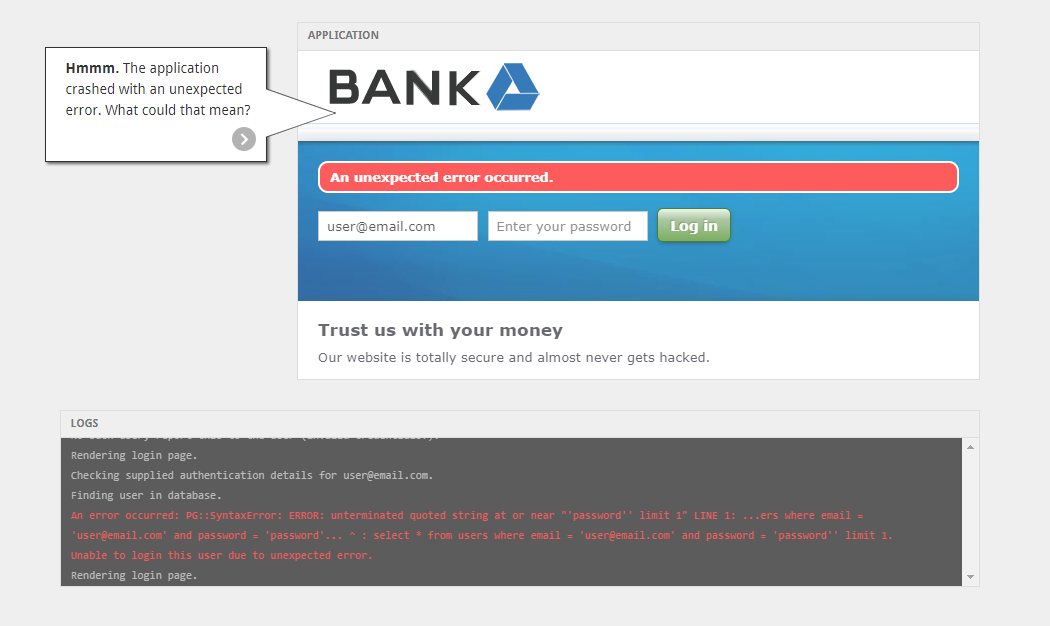
Password: password



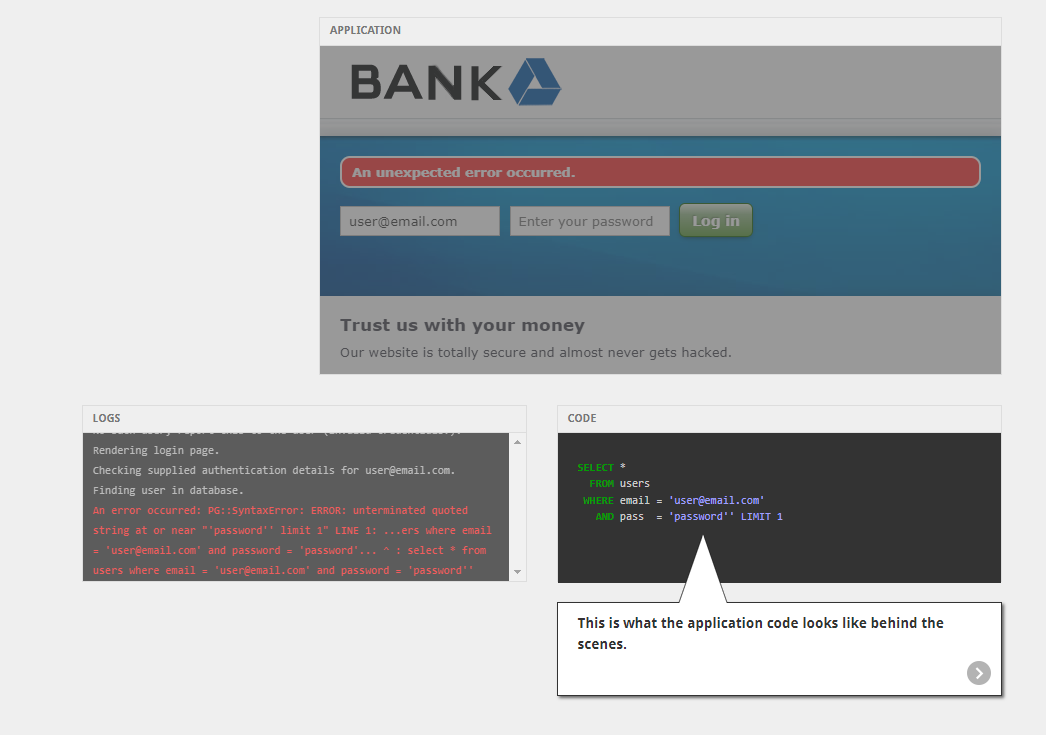
Добре, пароль не підходить, давайте спробуємо додати до паролю символ лапок.



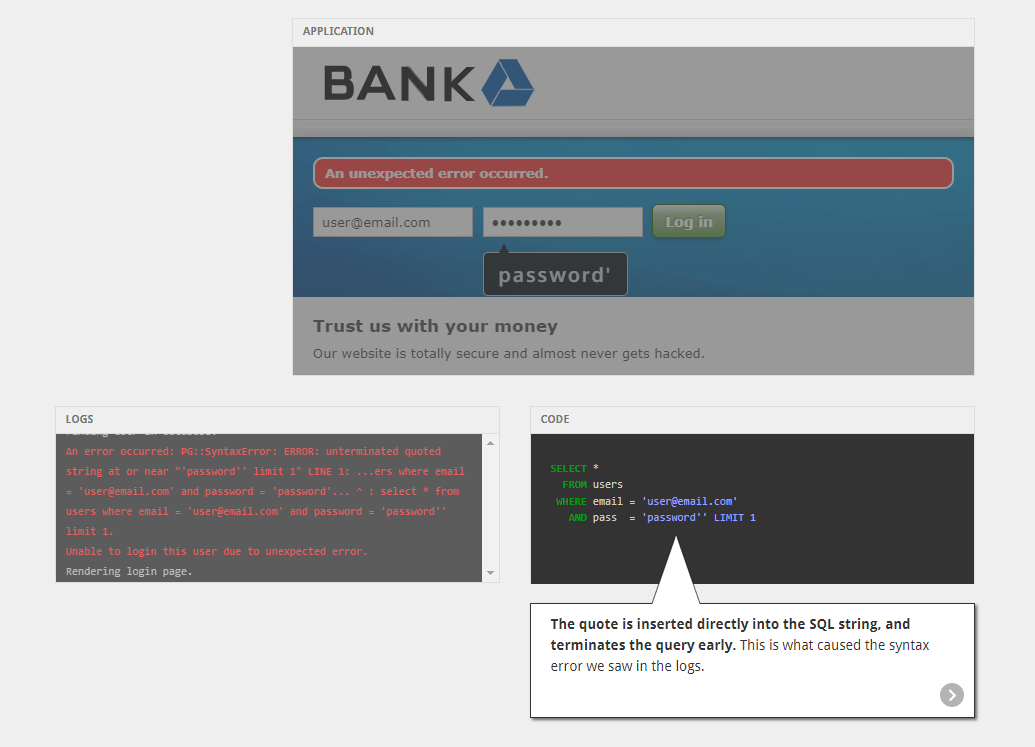
Додаток вилетів через непередбачувану помилку, та намагаємось зрозуміти звідки вона взялася. З логів бачимо, що це синтаксична помилка, що символ лапок щось несподівано зіпсував.



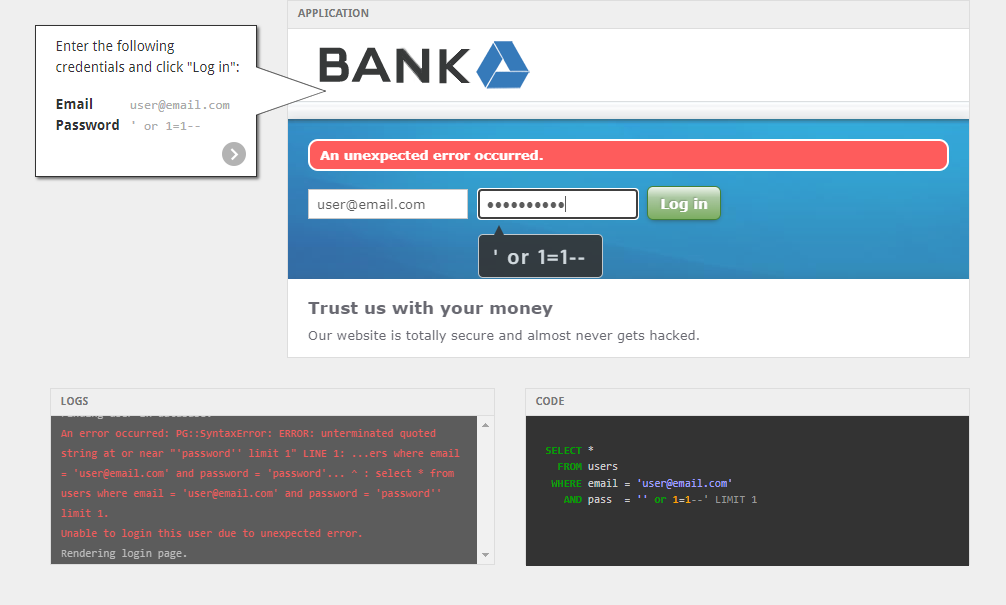
Ми бачимо як працює код та можемо спробувати ввести той самий пароль ще раз.



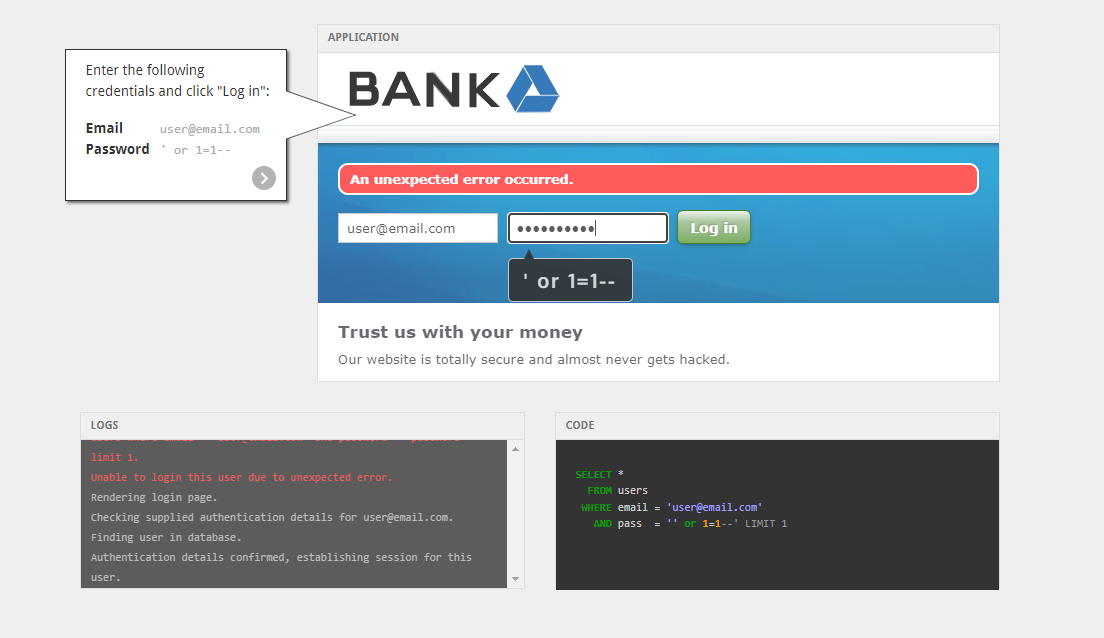
Цей символ лапок вставляється напряму в SQL рядок та зумовлює передчасне завершення запиту. Саме це й викликає синтаксичну помилку, яку ми бачили в логах. Це говорить нам про те, що додаток вразливий до SQL-ін’єкцій.



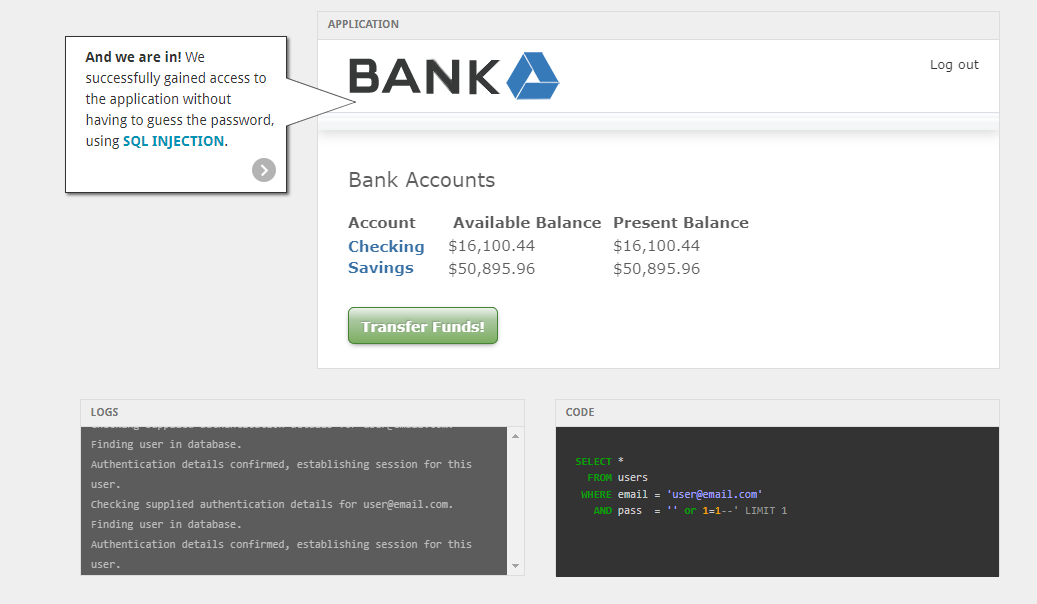
Вводимо нові облікові дані та натискаємо на логін.



Виконаною синтаксичною маніпуляцією ми вставили булеве значення 1 та закоментували частину коду.



Таким чином при проігнорували авторизацію та увійшли до акаунту без вводу пароля.



**Теоретичні відомості**

SQL-ін’єкція – тип ін’єкційних атак. Ін’єкційні атаки відбуваються, коли зловмисник відправляє шкідливі вхідні дані, що внаслідок змушує додаток виконувати непередбачувані дії. Через широке поширення використання SQL баз даних, SQL-ін’єкція є одним з найбільш поширених типів атак в Інтернеті.

**Ризики:**

Що найбільш страшне може відбутися, якщо атакували SQL-ін’єкцією?

В минулому хакери використовували такі атаки для:

1. Вилучення конфіденційної інформації, наприклад, номерів соціального страхування або даних кредитних карт.
2. Отримання аутентифікаційних даних користувачів, що зареєстровані на цьому сайті аби ці логіни можна було використовувати для атак на інших сайтах.
3. Видаляти дані або знищувати таблиці, пошкоджуючи базу даних та роблячи сайт непридатним до користування.
4. Впровадити додатковий шкідливий код, який буде виконуватися при відвідуванні сайту користувачами.

Атаки з використанням SQL-ін’єкції вражаюче розповсюджені. Такі

крупні компанії як Yahoo та Sony були зламані. В інших випадках хакерські групи націлювались на конкретні додатки чи писали сценарії, що були призначені для збирання даних аутентифікації.

**Захист:**

1. Параметризовані твердження

Мови програмування спілкуються з базами даних SQL за допомогою драйверів бази даних. Драйвер дозволяє програмі створювати та виконувати SQL-запити до бази даних, виймаючи та маніпулюючи даними у міру необхідності. Параметризовані твердження гарантують, що параметри (тобто вхідні дані), що передаються SQL-запити, обробляються безпечним чином.

Наприклад, безпечний спосіб виконання SQL-запиту JDBC з використанням параметризованого оператора виглядає наступним чином:

// Connect to the database.

Connection conn = DriverManager.getConnection(URL, USER, PASS);

// Construct the SQL statement we want to run, specifying the parameter.

String sql = "SELECT \* FROM users WHERE email = ?";

// Generate a prepared statement with the placeholder parameter.

PreparedStatement stmt = conn.prepareStatement(sql);

// Bind email value into the statement at parameter index 1.

stmt.setString(1, email);

// Run the query...

ResultSet results = stmt.executeQuery(sql);

**while** (results.next())

{

// ...do something with the data returned.

}

Протиставте це явній побудові рядка SQL, що дуже, дуже небезпечно:

// The user we want to find.

String email = "user@email.com";

// Connect to the database.

Connection conn = DriverManager.getConnection(URL, USER, PASS);

Statement stmt = conn.createStatement();

// Bad, bad news! Don't construct the query with string concatenation.

String sql = "SELECT \* FROM users WHERE email = '" + email + "'";

// I have a bad feeling about this...

ResultSet results = stmt.executeQuery(sql);

**while** (results.next()) {

// ...oh look, we got hacked.

}

Ключова відмінність полягає в даних, що передаються в метод executeQuery(...). У першому випадку параметризований рядок та параметри передаються в базу даних окремо, що дозволяє драйверу правильно їх інтерпретувати. У другому випадку повний SQL-запит створюється до виклику драйвера, що означає, що ми є вразливими для зловмисно створених параметрів.

**Ви завжди повинні використовувати параметризовані оператори там, де це можливо, вони є вашим захистом номер один від SQL-ін'єкцій.**

1. Об'єктно-реляційне відображення

Багато команд розробників вважають за краще використовувати фреймворки об'єктно-реляційного відображення (ORM), щоб зробити переведення наборів результатів SQL в об'єкти коду більш плавним. Інструменти ORM часто означають, що розробникам рідко доводиться писати SQL-запити у своєму коді – і ці інструменти, на щастя, використовують параметризовані запити під капотом.

Найвідомішим ORM, ймовірно, є фреймворк Active Record у Ruby on Rails. Отримання даних із бази даних за допомогою Active Record виглядає так:

**def** current\_user(email)

# The 'User' object is an Active Record object, that has find methods

# auto-magically generated by Rails.

User.find\_by\_email(email)

**end**

Такий код захищений від атак SQL Injection.

Однак, використання ORM не робить вас автоматично захищеним від SQL-ін'єкцій. Багато ORM-фреймворків дозволяють створювати SQL-запити або їх фрагменти, коли необхідно виконати складніші операції з базою даних. Наприклад, наступний код на мові Ruby вразливий для ін'єкційних атак:

**def** current\_user(email)

# This code would be vulnerable to a maliciously crafted email parameter.

User.where("email = '" + email + "'")

**end**

**Як загальне правило: якщо ви виявили, що пишете SQL-запити шляхом конкатенації рядків, дуже добре подумайте про те, що ви робите.**

1. Екранування вхідних параметрів

Якщо ви не можете використовувати параметризовані оператори або бібліотеку, яка пише SQL за вас, наступний найкращий підхід – забезпечити правильне екранування спеціальних рядкових символів у вхідних параметрах.

Ін'єкційні атаки часто ґрунтуються на тому, що зловмисник може створити вхідні дані, які передчасно закриють ряд аргументів, в якому вони з'являються в операторі SQL. (Саме тому у спробах атак SQL-ін'єкцій часто зустрічаються символи ' або ").

Мови програмування мають стандартні способи опису рядків, що містять у собі лапки - SQL у цьому відношенні нічим не відрізняється. Як правило, подвоєння символу лапок - заміна 'на' - означає "розглядати цю лапку як частину рядка, а не як кінець рядка".

Екранування символьних знаків - простий спосіб захисту від більшості атак SQL-ін'єкцій, і багато мов є стандартні функції для цього. Однак цей підхід має кілька недоліків:

Вам потрібно бути дуже обережним, щоб екранувати символи у вашій кодовій базі, де будується SQL-запит.

Не всі ін'єкційні атаки засновані на зловживанні символами лапок. Наприклад, коли в SQL-запиті очікується числовий ідентифікатор, символи лапок не потрібні. Наступний код все ще вразливий для атак ін'єкцій, незалежно від того, як багато ви граєте із символами лапок:

**def** current\_user(id)

User.where("id = " + id)

**End**

1. Санування входів

Санування даних, що вводяться, є гарною практикою для всіх додатків. У нашому прикладі злому користувач ввів пароль у вигляді 'або 1=1--, що виглядає досить підозріло як вибір пароля.

Розробники завжди повинні докладати зусиль, щоб відхиляти дані, які виглядають підозріло, але при цьому слідкувати за тим, щоб випадково не покарати легітимних користувачів. Наприклад, ваша програма може очищати параметри, що передаються в GET-і POST-запитах, такими способами:

Переконайтеся, що поля, що вводяться, наприклад, адреси електронної пошти, відповідають регулярному виразу.

Переконайтеся, що цифри або літерно-цифрові поля не містять символів.

Відхилити (або прибрати) пробіли та символи нового рядка, якщо вони не підходять.

Валідація на стороні клієнта (тобто JavaScript) корисна для надання користувачеві негайного зворотного зв'язку при заповненні форми, але не є захистом від серйозного хакера. Більшість спроб злому здійснюється за допомогою скриптів, а не самого браузера.

**Висновки**

У ході проходження курсу я ознайомився з поняттям SQL-ін’єкція та з найпростішим прикладом її реалізації. Також я вивчив теоретичні відомості щодо ризиків, які завдають атаки з використанням SQL-ін’єкції та засобів запобігання такого роду атакам.