



DOP: / /2023

DOS: / /2023

Experiment No:

Title: Insecure logging and Client-side injection.

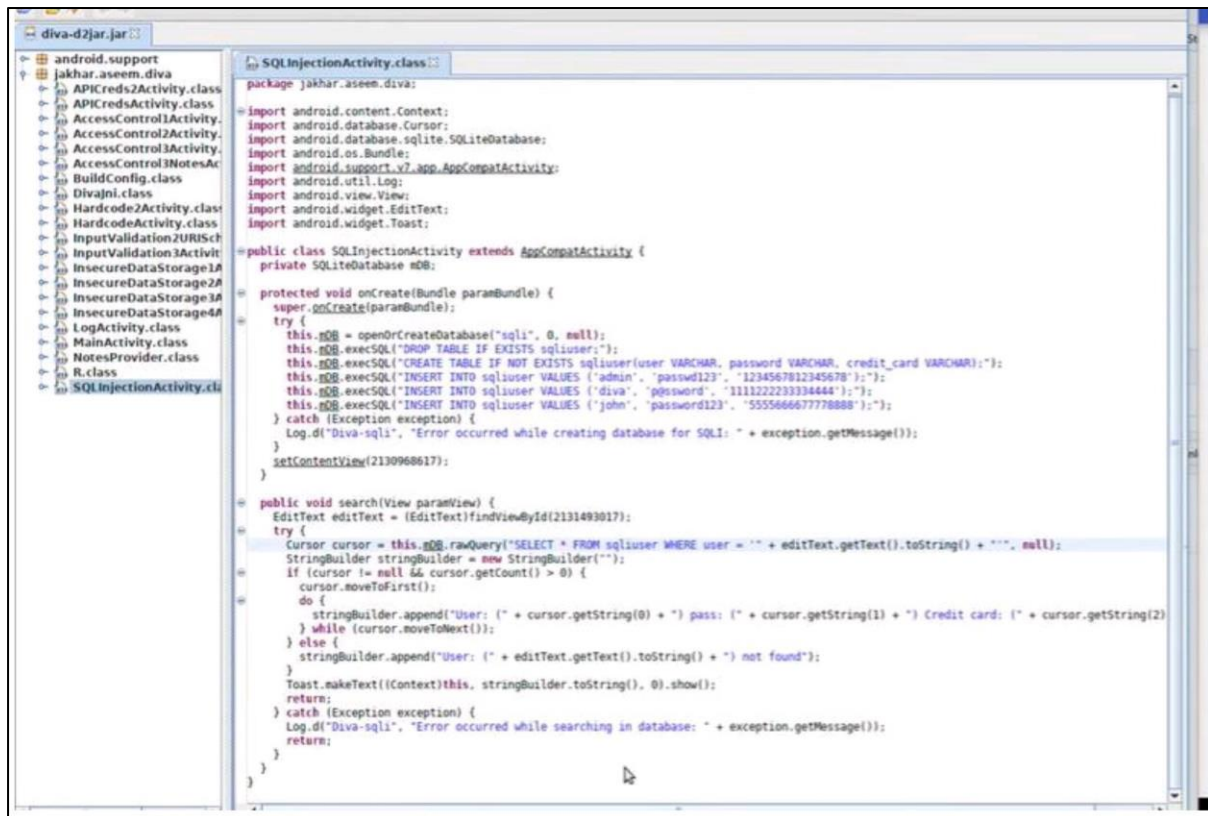
Theory:

Insecure logging refers to the practice of logging sensitive or confidential information without appropriate security measures in place, such as encryption or access controls. This can include personal information like usernames, passwords, credit card numbers, and other sensitive data that should be protected from unauthorized access.

Client-side injection, on the other hand, refers to a type of security vulnerability that occurs when untrusted data is passed into a web application and executed on the client-side. This can lead to a range of attacks, such as cross-site scripting (XSS), which allows an attacker to execute malicious code within a victim's web browser.

In combination, insecure logging and client-side injection can create a dangerous situation for both users and the organizations that operate the affected web applications. If sensitive information is logged without proper protection, it can be stolen by attackers who exploit client-side injection vulnerabilities. Additionally, client-side injection attacks can be used to bypass security controls and gain unauthorized access to sensitive data, which can then be logged by the application in an insecure manner.

To mitigate these risks, organizations should implement secure logging practices, such as encrypting sensitive data and limiting access to logs only to authorized personnel. Additionally, web applications should be designed with security in mind and tested regularly for vulnerabilities like client-side injection. Developers can use techniques like input validation and output encoding to prevent untrusted data from being executed on the client-side.



```

package jakhar.aseem.diva;

import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

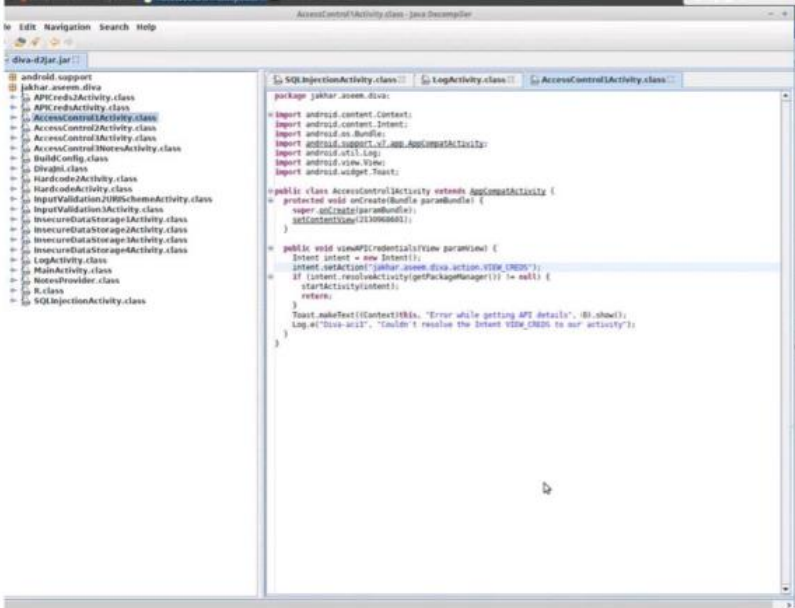
public class SQLInjectionActivity extends AppCompatActivity {
    private SQLiteDatabase mDB;

    protected void onCreate(Bundle paramBundle) {
        super.onCreate(paramBundle);
        try {
            this.mDB = openOrCreateDatabase("sql", 0, null);
            this.mDB.execSQL("DROP TABLE IF EXISTS sqluser;");
            this.mDB.execSQL("CREATE TABLE IF NOT EXISTS sqluser(user VARCHAR, password VARCHAR, credit_card VARCHAR);");
            this.mDB.execSQL("INSERT INTO sqluser VALUES ('admin', 'password123', '1234567812345678');");
            this.mDB.execSQL("INSERT INTO sqluser VALUES ('diva', 'password', '1111222233334444');");
            this.mDB.execSQL("INSERT INTO sqluser VALUES ('john', 'password123', '5555666677778888');");
        } catch (Exception exception) {
            Log.d("Divs-sql", "Error occurred while creating database for SQL: " + exception.getMessage());
        }
        setContentView(2130968617);
    }

    public void search(View paramView) {
        EditText editText = (EditText)findViewById(2131493017);
        try {
            Cursor cursor = this.mDB.rawQuery("SELECT * FROM sqluser WHERE user = '" + editText.getText().toString() + "'", null);
            StringBuilder stringBuilder = new StringBuilder("");
            if (cursor != null && cursor.getCount() > 0) {
                cursor.moveToFirst();
                do {
                    stringBuilder.append("User: (" + cursor.getString(0) + ") pass: (" + cursor.getString(1) + ") Credit card: (" + cursor.getString(2) + ")");
                    while (cursor.moveToNext());
                } else {
                    stringBuilder.append("User: (" + editText.getText().toString() + ") not found");
                }
                Toast.makeText((Context)this, stringBuilder.toString(), 0).show();
                return;
            } catch (Exception exception) {
                Log.d("Divs-sql", "Error occurred while searching in database: " + exception.getMessage());
                return;
            }
        }
    }
}

```





7. Input Validation Issues - Part 1

Objective: Try to access all user data without knowing any user name. There are three users by default and your task is to output data of all the three users with a single malicious search.

Hint: Improper or no input validation issue arise when the input is not filtered or validated before using it. When developing components that take input from outside, always validate it. For ease of testing there are three users already present in the database, for example one of them is admin, you can try searching for admin to test the output.

test' or '1'='1

SEARCH

Conclusion: - Thus we have successfully studied insecure logging and client -side injection.