Android-InsecureBankv2 Reverse Analysis Report

1. Preface

> This project serves as both a home task for an interview and an opportunity for learning Android reverse engineering. The analysis includes static and dynamic analysis of the Android application "InsecureBankv2" hosted on GitHub.
>
> - Task:
>
> **Test Overview and Objective**
> Reverse engineers choose either Android APK or iOS IPA to extract undocumented API endpoints and authentication logic, and demonstrate the ability to bypass a basic anti-debugging measure.
>
> **Deliverables:**
> 1. Written report detailing the approach, tools used, and findings
> 2. Extracted API endpoints and authentication flow (e.g., how tokens are generated)
> 3. Code/scripts used for dynamic analysis, traffic interception, or automation
> 4. (Optional) A brief video walkthrough (5-10min) explaining the process
>
> **Test Details:**
> 1. Static Analysis
>   a. Decompile the APK/IPA
>   b. Identify Key classes/methods related to network communication and authentication
>   c. Document the API endpoints and parameters used
> 2. Dynamic Analysis
>   a. Setup a proxy to intercept app traffic
>   b. Demonstrate ability to hook or patch code to bypass a simple anti-debugging check
>   c. Extract authentication tokens or session data
> 3. Bonus
>   a. Identify and explain any obfuscation or encryption used in the app
>   b. Suggest improvements to app security based on findings
>
> **Evaluation Criteria**
> 1. Technical accuracy and depth of analysis
> 2. Clarity and thoroughness of documentation
> 3. Use of appropriate tools
> 4. Creativity and problem-solving approach
>
> - Target: https://github.com/dineshshetty/Android-InsecureBankv2
> - Tools:
>   - chatgpt !!!
>   - apktool
>   - jadx
>   - Android Studio
>   - BurpSuite
>   - OS: Apple M4 (Macbook Air)
>   - Cmdline tool: iTerm + oh-my-zsh !!!

> - Process:
>   - decide target : Android + InsecureBank
>   - Static Analysis: jadx + chatgpt
>   - Dynamic Analysis: adb + apktool + BurpSuite
>   - Bonus summary
>   - create GitHub repository and upload
> - Time: 2 days

2. Static Analysis

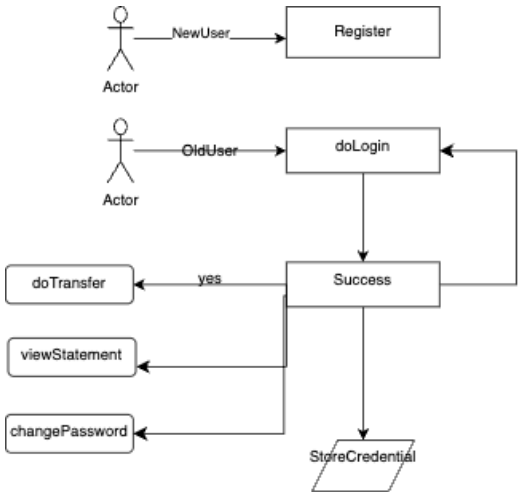
2.1 Tools installation

- Jadx: `brew install jadx`

2.2 Download target apk

- `git clone git@github.com:dineshshetty/Android-InsecureBankv2.git`

2.3 Sequence of login



- new user: createUser
- oldUser: filldata from local -> perform login -> check if success
  - Success -> save Creds(user, pass) ->PostLogin -> (doTransfer / viewStatement / changePassword)
    - api: http://ip:port/dotransfer
      - Post
      - username+password+from_acc+to_acc+amount
    - api: http://ip:port/getaccounts
      - Post
      - username + password

        - api: http://ip:port/changepassword
            - Post
            - username + new password
    - Fail -> WrongLogin -> login

2.4 Login Action

- API:
    - http://ip:port/login
        - post
        - username + password
    - http://ip:port/devlogin
        - post
        - username + password
- **Problems**:
    - Using HTTP, easy to be intercepted by man-in-the-middle (should use HTTPS).

```
String responseString = null;
String serverip = "";
String serverport = "";
String protocol = "http://";
```

    - Using `SharedPreferences` for storing credentials in an insecure manner (passwords use AES encryption with hardcoded key).
    - Logging usernames and passwords with `android.util.Log` (easily exposed).
    - **Recommendations**:
        - Do not store credentials locally.
        - Use Android Keystore and AES-GCM.
        - Use HTTPS for secure communication.

2.5 Change Password Vulnerability

- No validation of the current password when changing the password.
- New password sent via SMS, easily intercepted.
- Password transmitted in plaintext, should be hashed which is not reversible. Shouldn't use crypto, because crypto value could be decrypted.

3. Dynamic Analysis

> The InsecureBank APK version is too old, so it couldn't be installed on Android Studio. Therefore, I installed cmdline-tools and used the `sdkmanager` command line to create an Android emulator with a lower API version (API 22). However, it still failed because the Apple M4 chip does not support Frida. Fortunately, the app itself uses the HTTP protocol. (In the future, I will use other APKs to study HTTPS bypass techniques.)

3.1 Emulator Setup

- Install ARM64-compatible Android 6.0 system image:

```
sdkmanager "system-images;android-23;google_apis;arm64-v8a"
```

- Create and launch an emulator:

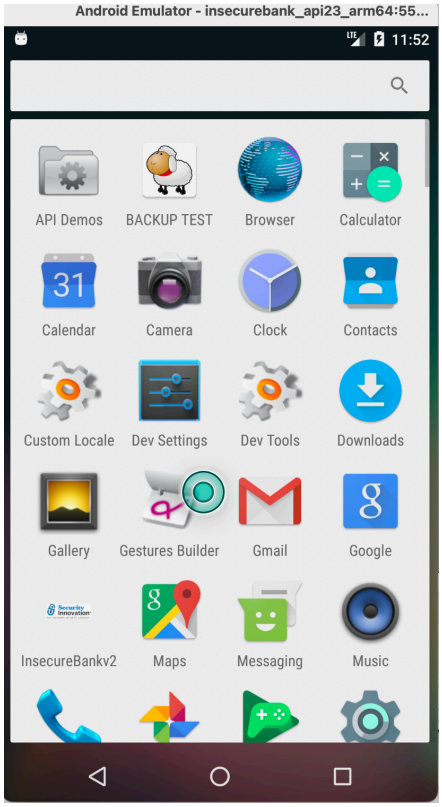
```
avdmanager create avd -n insecurebank_api23_arm64 -k "system-images;android-23;google_apis;arm64-v8a" --device "pixel" --force
```

- Start Emulator (use `adb devices` check weather emulator started)

```
emulator -avd insecurebank_api23_arm64
```

3.2 Install InsecureBankv2.apk in Emulator

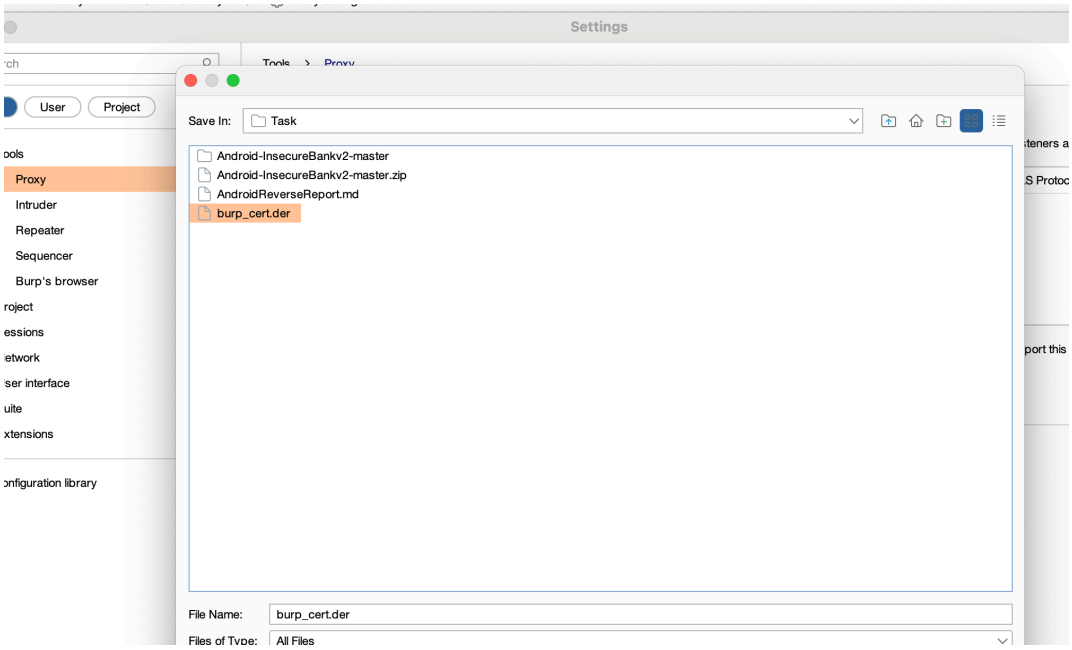
```
adb install ~/Desktop/Task/Android-InsecureBankv2-master/InsecureBankv2.apk
```



3.3 Burp Suite Configuration

- Install Burp certificate to the emulator:

```
adb root
adb remount
adb push burp_cert.der /sdcard/
```



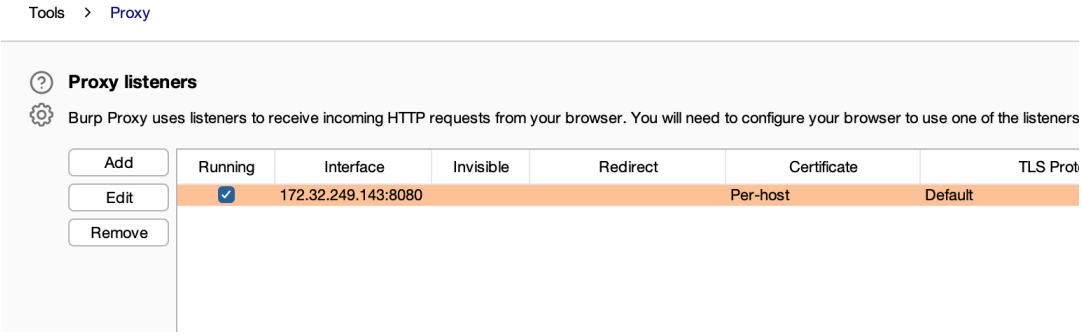
3.4 Frida Installation Issues

- Due to Apple M4 chip compatibility issues, Frida installation failed. Used proxy for interception. And found that InsecureBank use HTTP so don't need frida anymore
-

```
    public static final String MYPREFS = "mySharedPreferences";
    String password;
    BufferedReader reader;
    String rememberme_password;
    String rememberme_username;
    String result;
    SharedPreferences serverDetails;
    String superSecurePassword;
    String username;
    String responseString = null;
    String serverip = "";
    String serverport = "";
    String protocol = "http://";
```

3.5 API Testing and Findings

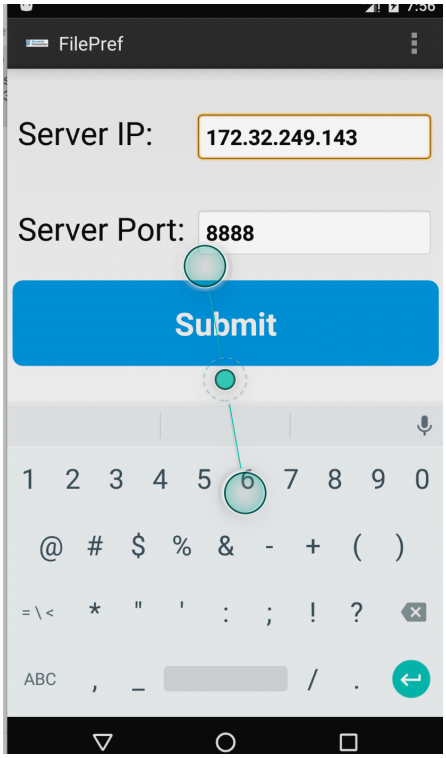
- Burp configuration



- Emulator proxy configuration

```
adb shell settings put global http_proxy 172.32.249.143:8080
```

- **Attention**！ Previously, the proxy address was configured as 10.0.2.2:8888, which led to many detours. In fact, Burp should listen on this address, and the emulator should be configured to use this address as the proxy. Do not set it to 10.0.2.2, otherwise, Burp will not forward requests from the local machine to Flask.
- Setup server ip port at insecureBank APP:



- start backend server:

```
cd /Users/keeplook4ever/Desktop/Task/Android-InsecureBankv2-master/AndroLabServer
conda activate
pip install -r requirements.txt
python app.py
```

- error, install py27 env:

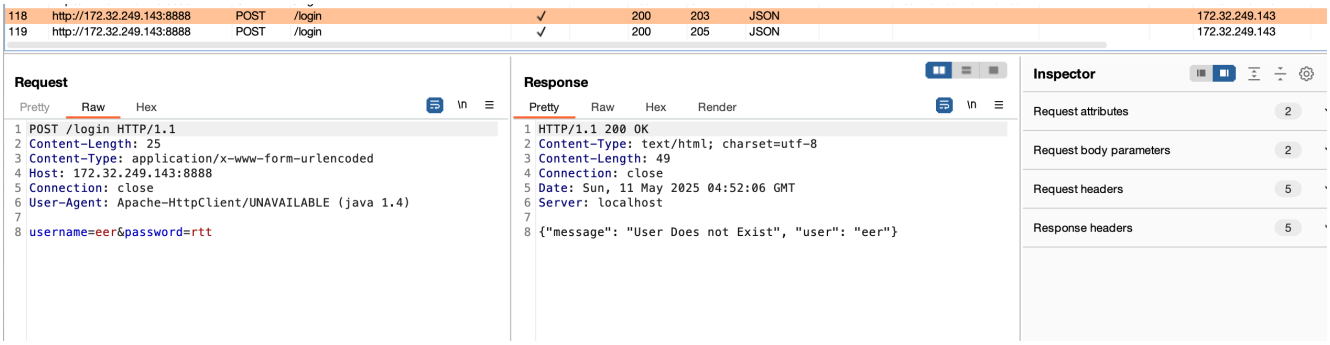
```
CONDA_SUBDIR=osx-64 conda create -n py27 python=2.7
CONDA_SUBDIR=osx-64 conda activate py27
pip install -r requirements.txt
python app.py
```

```
Successfully built sqlalchemy simplejson web.py scandir
Installing collected packages: MarkupSafe, Jinja2, Werkzeug, click, itsdangerous, flask, configparser, contextlib2, typing, scandir, six, pathlib2, zipp, importlib-metadata, sqlalchemy, simplejson, more-itertools, selectors2, backports.functools-lru-cache, jaraco.functools, cheroot, web.py, zc.lockfile, pytz, tempora, portend, cherrypy
Successfully installed Jinja2-2.11.3 MarkupSafe-1.1.1 Werkzeug-1.0.1 backports.functools-lru-cache-1.6.6 cheroot-8.6.0 cherrypy-17.4.2 click-7.1.2 configparser-4.0.2 contextlib2-0.6.0.post1 flask-1.1.4 importlib-metadata-2.1.3 itsdangerous-1.1.0 jaraco.functools-2.0 more-itertools-5.0.0 pathlib2-2.3.7.post1 portend-2.6 pytz-2025.2 scandir-1.10.0 selectors2-2.0.2 simplejson-3.20.1 six-1.17.0 sqlalchemy-1.4.54 tempora-1.14.1 typing-3.10.0.0 web.py-0.51 zc.lockfile-2.0 zipp-1.2.0
(py27)  keeplook4ever  ~/Desktop/Task/Android-InsecureBankv2-master/AndroLabServer  python app.py
The server is hosted on port: 8888
```

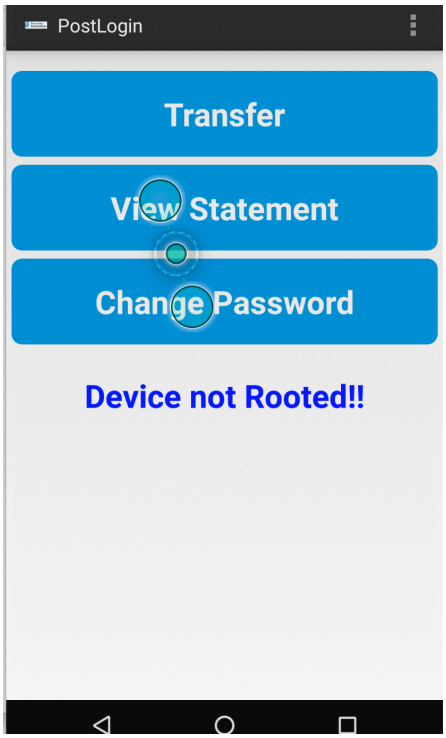
3.6 Login Traffic Capture

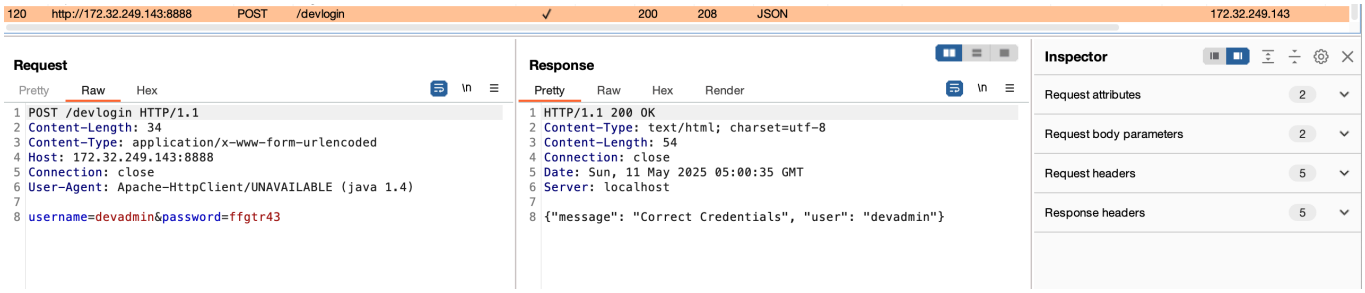
- username + password: eer/xxxx

```
(py27)  keeplook4ever  ~/Desktop/Task/Android-InsecureBankv2-master/AndroLabServer  python app.py
The server is hosted on port: 8888
u= None
{"message": "User Does not Exist", "user": "eer"}
```

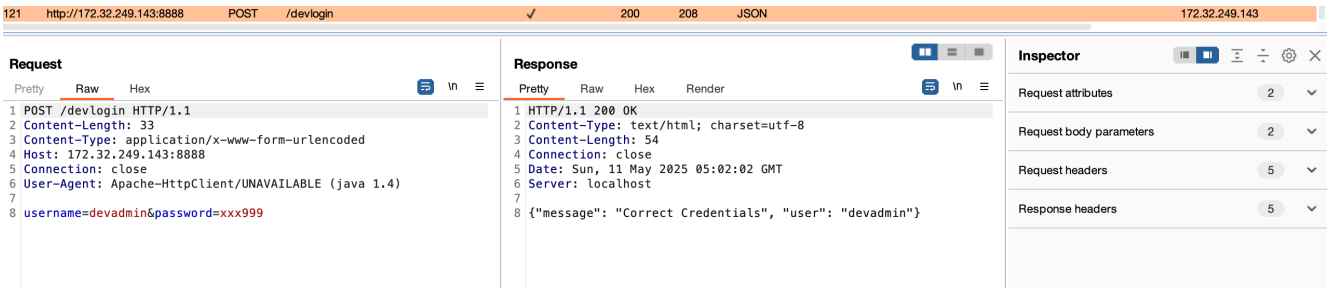
- 

- Since the `devadmin` username was discovered during static code review, I tried logging in with this username to see if it's a backdoor: It successfully logged in without any issue.



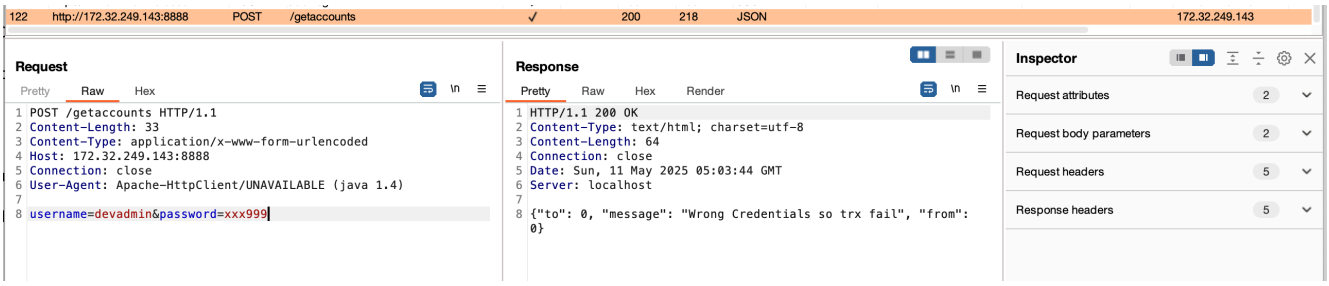


- Further verification: The password check should not exist. I tried changing the password to: devadmin/xxx999, and as expected, I was able to log in successfully, confirming that the `devadmin` account is a developer/test account with no password check.

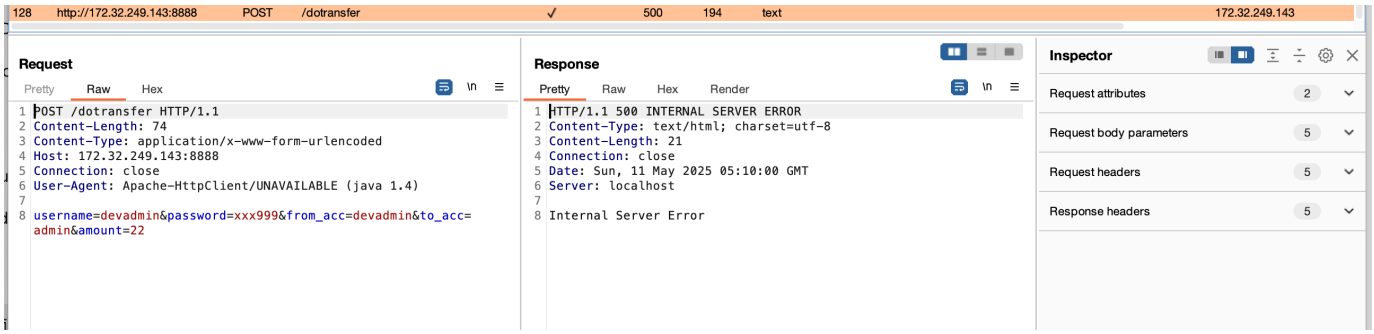


3.7 Transfer API Traffic Capture:

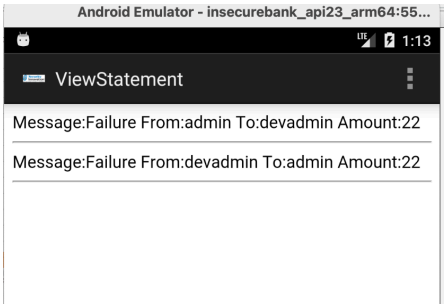
- **Discovery**: The previously static-reviewed `/getaccounts` API allows brute-forcing to guess platform user account credentials.



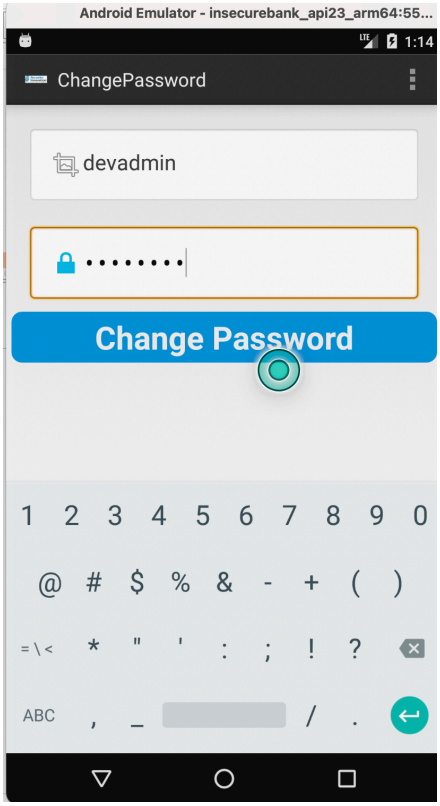
- **`dotransfer` API**: This API passes `username+password` to check user permissions, but if a hacker obtains any user's account and password, they can perform any transfer by setting `to_acc=hacker`. Since there's a backend error, it could not be tested.



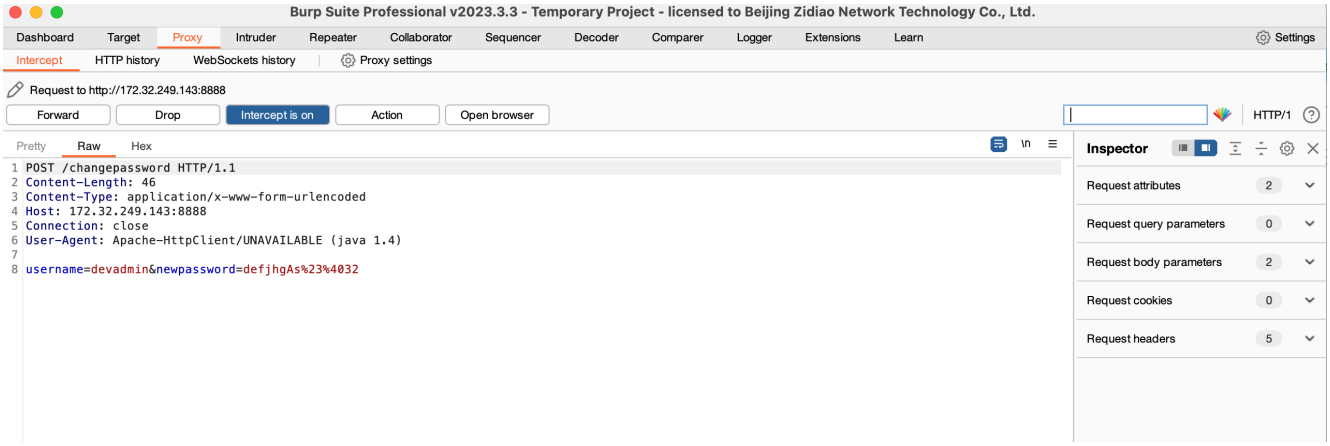
3.8 viewstatement：



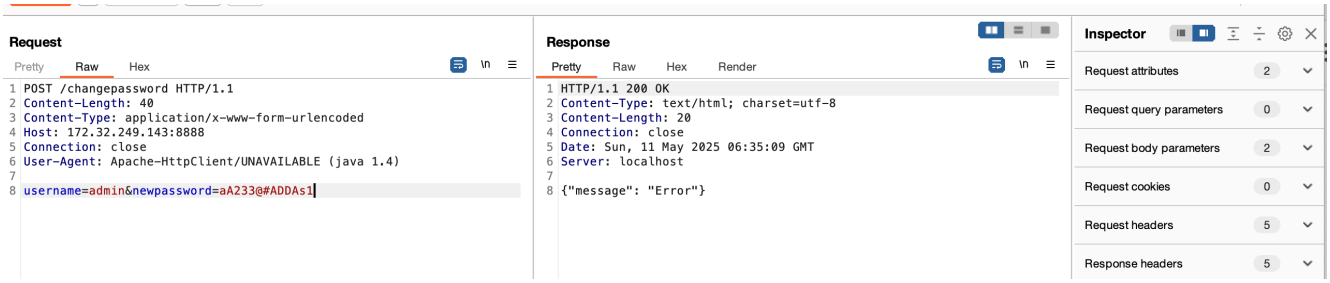
3.9 Test `changepasswd` API:



- **Interception and Replay**:



- **Replayed Successfully**: As shown, the request replayed successfully (response code: 200, message: Error, which is a backend error and can be ignored). This allows changing any existing user's password, a serious vulnerability.



3.10 Rebuilt SDK for Admin Access

> In static code analysis, I found that "button_CreateUser" is displayed only when `"R.string.is_admin"` is set to "yes". I attempted to bypass this by modifying the code:

```
@Override // android.app.Activity
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_log_main);
    String mess = getResources().getString(R.string.is_admin);
    if (mess.equals("no")) {
        View button_CreateUser = findViewById(R.id.button_CreateUser);
        button_CreateUser.setVisibility(8);
    }
    this.login_buttons = (Button) findViewById(R.id.login_button);
    this.login_buttons.setOnClickListener(new View.OnClickListener() { // from class: co
        @Override // android.view.View.OnClickListener
        public void onClick(View v) {
            LoginActivity.this.performlogin();
        }
    });
    this.createuser_buttons = (Button) findViewById(R.id.button_CreateUser);
    this.createuser_buttons.setOnClickListener(new View.OnClickListener() { // from clas
        @Override // android.view.View.OnClickListener
        public void onClick(View v) {
            LoginActivity.this.createUser();
        }
    });
    this.fillData_button = (Button) findViewById(R.id.fill_data);
```

- Install APKTool:

```
brew install apktool
```

- Decompile APK:

```
apktool d ~/Desktop/Task/Android-InsecureBankv2-master/InsecureBankv2.apk -o
insecurebank_dec
```

- Modify `res/values/strings.xml` to set `is_admin` to "yes":
- Rebuild and sign the APK:

```
apktool b InsecureBank_decoded -o InsecureBankv2_admin.apk
apksigner sign --ks debug.keystore --ks-key-alias androiddebugkey --ks-pass
pass:android InsecureBankv2_admin.apk
```

- Since `apksigner` was not found, I used the command `find ~/Library/Android/sdk/build-tools -name apksigner` to locate the `apksigner` tool:

  ```
  /Users/keeplook4ever/Library/Android/sdk/build-tools/35.0.1/apksigner
  /Users/keeplook4ever/Library/Android/sdk/build-tools/36.0.0/apksigner
  ```

- Then used the absolute path to sign:

  ```
  /Users/keeplook4ever/Library/Android/sdk/build-tools/35.0.1/apksigner sign \
    --ks debug.keystore \
    --ks-key-alias androiddebugkey \
    --ks-pass pass:android \
    InsecureBankv2_admin.apk
  ```

- Generate a new `debug.keystore` if it is missing:

  ```
  keytool -genkey -v -keystore debug.keystore \
    -storepass android -alias androiddebugkey \
    -keypass android -keyalg RSA -keysize 2048 -validity 10000
  ```

- Install the signed APK on the emulator:

  ```
  adb install -r InsecureBankv2_admin.apk
  ```

  If there was an error, uninstall the previous APK and reinstall:

  ```
  adb uninstall com.android.insecurebankv2
  ```

- Successfully displayed the "Create User" button: Although the backend doesn't implement the functionality, this proves that the modified APK allows creating any account (a critical vulnerability, as only admins should have this access).

4. Bonus for Credentials and Access Control

4.1. Direct Exposure of Backend Database Address and Tables

The application defines a custom `ContentProvider` for tracking user information but does not set access control, allowing any app to query or tamper with the content. Testers can retrieve, delete, or forge user records via the `content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers` URI. Additionally, database fields are stored in plaintext without access control or audit logs. It is recommended to restrict export permissions and encrypt sensitive fields.

```
/* loaded from: classes.dex */
public class TrackUserContentProvider extends ContentProvider {
    static final String CREATE_DB_TABLE = " CREATE TABLE names (id INTEGER PRIMARY KEY AUTOINCREMENT,  name TEXT NOT NULL);";
    static final String DATABASE_NAME = "mydb";
    static final int DATABASE_VERSION = 1;
    static final String PROVIDER_NAME = "com.android.insecurebankv2.TrackUserContentProvider";
    static final String TABLE_NAME = "names";
    static final String name = "name";
    static final int uriCode = 1;
    private static HashMap<String, String> values;
    private SQLiteDatabase db;
    static final String URL = "content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers";
    static final Uri CONTENT_URI = Uri.parse(URL);
    static final UriMatcher uriMatcher = new UriMatcher(-1);

    static {
        uriMatcher.addURI(PROVIDER_NAME, "trackerusers", 1);
        uriMatcher.addURI(PROVIDER_NAME, "trackerusers/*", 1);
    }

    @Override // android.content.ContentProvider
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        switch (uriMatcher.match(uri)) {
            case 1:
                int count = this.db.delete(TABLE_NAME, selection, selectionArgs);
                getContext().getContentResolver().notifyChange(uri, null);
                return count;
            default:
                throw new IllegalArgumentException("Unknown URI " + uri);
        }
```

The code directly exposes the backend server database address and table names, making it easy to leak data. You can directly query users with adb, for example, finding users `jack` and `devadmin`:

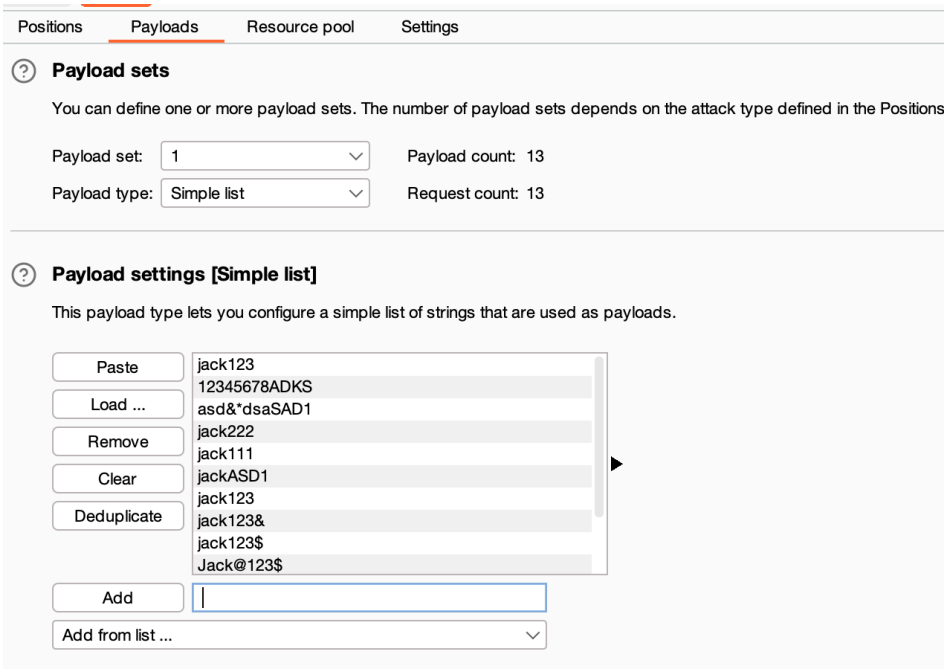
```
adb shell content query --uri
content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
```

```
(py27)  keeplook4ever  ~/Desktop/Task/Android-InsecureBankv2-master  adb shell content query --uri content://com.andro
id.insecurebankv2.TrackUserContentProvider/trackerusers

Row: 0 id=1, name=devadmin
Row: 1 id=2, name=jack
```

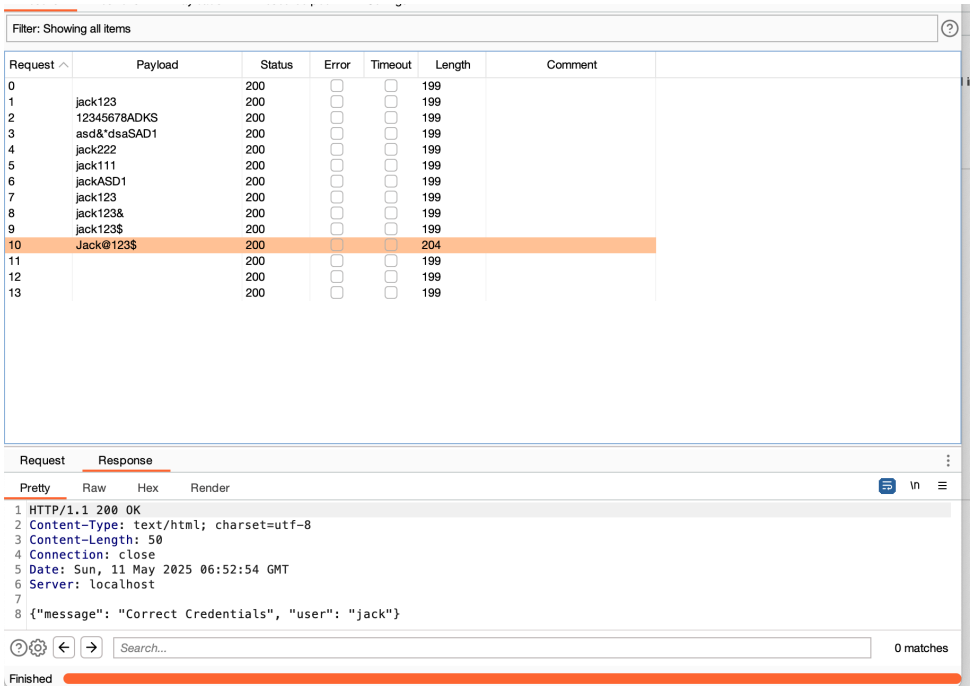
4.2. Login Brute-Force, No Session/Token Used in Login Records

For example, using `jack`'s username, brute-force password payloads can be constructed:



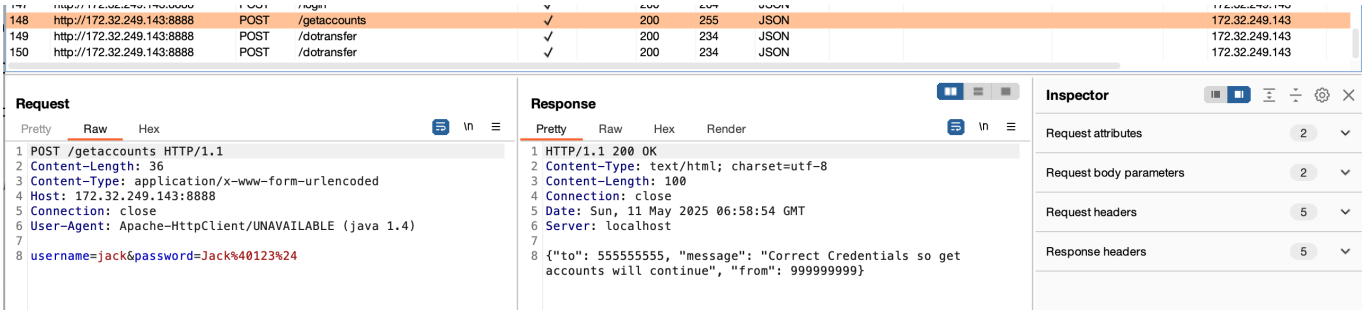
The result of the attack allows filtering out successful passwords by comparing response lengths.

Fix Recommendation: Limit the number of login attempts for the same device. For example, after 3 failed attempts within 5 minutes, prompt for CAPTCHA or block the current device/IP from attempting further logins. Account lockout should be avoided as it could lock normal user accounts when hackers attempt brute-forcing.

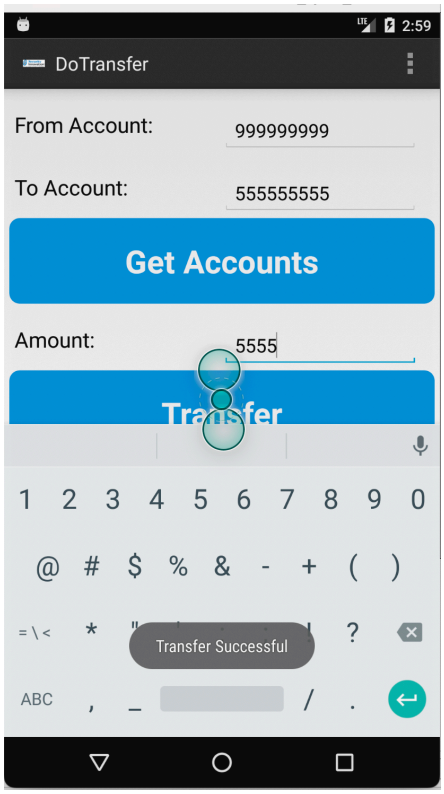


4.3. Transfer API Not Verified, Source Account Not Checked

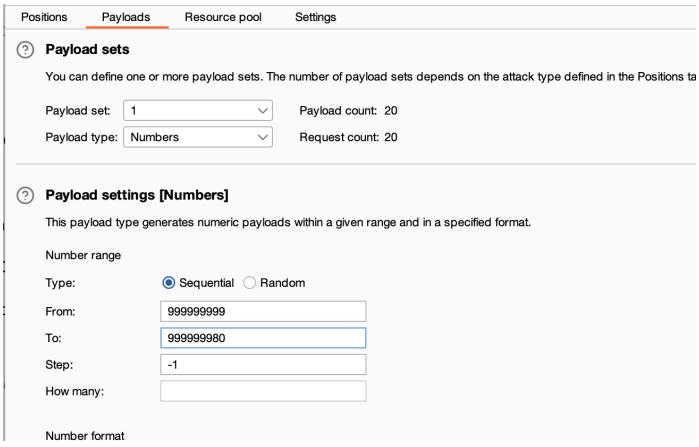
Input: `from account: admin`, `to account: jack`, current logged-in user is `jack` with account number `555555555`, and `admin` has account `999999999`.

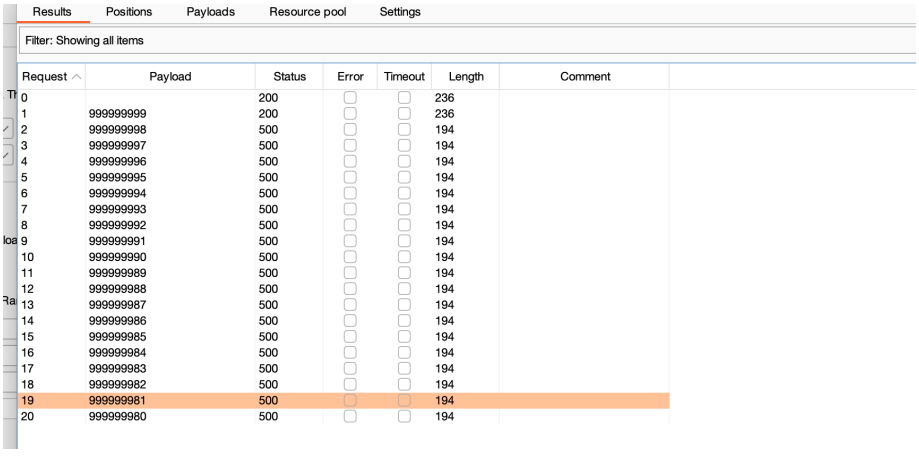


Tested the `transfer` API and successfully transferred funds.



This allowed unauthorized transfers, and funds could be transferred from all accounts to `jack` by exploiting the vulnerability. Testing with 10000 over 20 accounts:





Request	Payload	Status	Error	Timeout	Length	Comment
0		200			236	
1	999999999	200			236	
2	999999998	500			194	
3	999999997	500			194	
4	999999996	500			194	
5	999999995	500			194	
6	999999994	500			194	
7	999999993	500			194	
8	999999992	500			194	
9	999999991	500			194	
10	999999990	500			194	
11	999999989	500			194	
12	999999988	500			194	
13	999999987	500			194	
14	999999986	500			194	
15	999999985	500			194	
16	999999984	500			194	
17	999999983	500			194	
18	999999982	500			194	
19	999999981	500			194	
20	999999980	500			194	

Only `999999999` was successful, indicating other account IDs did not exist in the payload. More attempts were not made due to time constraints.

4.4. Password Encryption Key Hardcoded, Local Storage and Logging

```
/* loaded from: classes.dex */
public class CryptoClass {
    String base64Text;
    byte[] cipherData;
    String cipherText;
    String plainText;
    String key = "This is the super secret key 123";
    byte[] ivBytes = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    public static byte[] aes256encrypt(byte[] ivBytes, byte[] keyBytes, byte[] textBytes) throws UnsupportedEncodingException, NoSuchAlgorithmE
        AlgorithmParameterSpec ivSpec = new IvParameterSpec(ivBytes);
        SecretKeySpec newKey = new SecretKeySpec(keyBytes, "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(1, newKey, ivSpec);
        return cipher.doFinal(textBytes);
    }

    public static byte[] aes256decrypt(byte[] ivBytes, byte[] keyBytes, byte[] textBytes) throws UnsupportedEncodingException, NoSuchAlgorithmE
        AlgorithmParameterSpec ivSpec = new IvParameterSpec(ivBytes);
        SecretKeySpec newKey = new SecretKeySpec(keyBytes, "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(2, newKey, ivSpec);
        return cipher.doFinal(textBytes);
    }
```

Code Issues:

- Hardcoded key, easy to reverse-engineer and leak, effectively like plaintext transmission.
- Static zero Initialization Vector (IV) in CBC mode leads to password leaks: identical plaintext passwords will always generate the same ciphertext.
- Unsafe AES mode: CBC, while strong, does not guarantee data integrity. AES-GCM should be used.

Example code to upgrade to AES-GCM with random IV:

```
SecureRandom random = new SecureRandom();
byte[] iv = new byte[12]; // GCM recommends 12-byte IV
random.nextBytes(iv);

SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
GCMParameterSpec gcmSpec = new GCMParameterSpec(128, iv);

Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
cipher.init(Cipher.ENCRYPT_MODE, keySpec, gcmSpec);

byte[] cipherText = cipher.doFinal(plainText.getBytes());
```

4.5 Get user and password with adb shell and decrypt for plaintext password

1. found username and password are stored at local android data using `sharedPreference`

```
import android.content.SharedPreferences;
```

```
private void saveCreds(String username, String password) throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidAlgorithmParameterException, IllegalBlockSizeException, BadPaddingException {
    SharedPreferences mySharedPreferences = DoLogin.this.getSharedPreferences("mySharedPreferences", 0);
    SharedPreferences.Editor editor = mySharedPreferences.edit();
    DoLogin.this.rememberme_username = username;
    DoLogin.this.rememberme_password = password;
    String base64Username = new String(Base64.encodeToString(DoLogin.this.rememberme_username.getBytes(), 4));
    CryptoClass crypt = new CryptoClass();
    DoLogin.this.superSecurePassword = crypt.aesEncryptedString(DoLogin.this.rememberme_password);
    editor.putString("EncryptedUsername", base64Username);
    editor.putString("superSecurePassword", DoLogin.this.superSecurePassword);
    editor.commit();
}
```

2. Run the following cmd to start shell debugging and locate the stored account and password data: (Make sure the emulator is running beforehand)

```
adb shell
run-as com.android.insecurebankv2
cd shared_prefs
cat mySharedPreferences.xml
```

```
keeplook4ever ~/Documents/FirstStepInReverse main adb shell
root@generic_arm64:/ # run-as insecurebankv2
run-as: Package 'insecurebankv2' is unknown
254|root@generic_arm64:/ # run-as com.android.insecurebankv2
d shared_prefs/                                                    <
root@generic_arm64:/data/data/com.android.insecurebankv2/shared_prefs $
cat mySharedPreferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="EncryptedUsername">ZGV2YWRtaW4=
    </string>
    <string name="superSecurePassword">i2soXgauVzM8iD/TBS8cbQ==
    </string>
</map>
root@generic_arm64:/data/data/com.android.insecurebankv2/shared_prefs $
```

3. Decrypt username and password
    1. EncryptedUsername value is `ZGV2YWRtaW4=`, known as using base64, so decode as `devadmin`
    2. superSecurePassword value is `i2soXgauVzM8iD/TBS8cbQ==`, known as using AES encrypt, the encrypt code of the app is shown as following:

```
/* loaded from: classes.dex */
public class CryptoClass {
    String base64Text;
    byte[] cipherData;
    String cipherText;
    String plainText;
    String key = "This is the super secret key 123";
    byte[] ivBytes = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

    public static byte[] aes256encrypt(byte[] ivBytes, byte[] keyBytes, byte[] textBytes) throws UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, Inval
        AlgorithmParameterSpec ivSpec = new IvParameterSpec(ivBytes);
        SecretKeySpec newKey = new SecretKeySpec(keyBytes, "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(1, newKey, ivSpec);
        return cipher.doFinal(textBytes);
    }

    public static byte[] aes256decrypt(byte[] ivBytes, byte[] keyBytes, byte[] textBytes) throws UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, Inval
        AlgorithmParameterSpec ivSpec = new IvParameterSpec(ivBytes);
        SecretKeySpec newKey = new SecretKeySpec(keyBytes, "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(2, newKey, ivSpec);
        return cipher.doFinal(textBytes);
    }

    public String aesDeccryptedString(String theString) throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidAlgorith
        byte[] keyBytes = this.key.getBytes("UTF-8");
        this.cipherData = aes256decrypt(this.ivBytes, keyBytes, Base64.decode(theString.getBytes("UTF-8"), 0));
        this.plainText = new String(this.cipherData, "UTF-8");
        return this.plainText;
    }

    public String aesEncryptedString(String theString) throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidAlgorithm
        byte[] keyBytes = this.key.getBytes("UTF-8");
        this.plainText = theString;
        this.cipherData = aes256encrypt(this.ivBytes, keyBytes, this.plainText.getBytes("UTF-8"));
        this.cipherText = Base64.encodeToString(this.cipherData, 0);
        return this.cipherText;
    }
}
```

4. Write decryption python codes :

```
from Crypto.Cipher import AES
```

```python
import base64

# 从 APK 中找到的 key 和 IV
key = "This is the super secret key 123".encode("utf-8")
iv = bytes([0] * 16)

# 密文 Base64 解码
cipher_b64 = "i2soXgauVzM8iD/TBS8cbQ=="
cipher_bytes = base64.b64decode(cipher_b64)

# 解密
cipher = AES.new(key, AES.MODE_CBC, iv)
plaintext = cipher.decrypt(cipher_bytes)

# 去除填充
pad_len = plaintext[-1]
plaintext = plaintext[:-pad_len]

print("Decrypted password:", plaintext.decode("utf-8"))
```

run and get the plaintext of password:

```
(base)  keeplook4ever  ~/Documents/FirstStepInReverse  main ±  python decrypt.py
Decrypted password: iioop
```

4.6. Misconfigured AndroidManifest.xml

1. `android:debuggable="true"` opens debug mode, allowing any USB debugging tool (adb shell, frida, re-framework) to attach.
   It should be changed to `android:debuggable="false"` or removed.
2. `android:allowBackup="true"` enables attackers to extract data via `adb backup`, which should be set to `android:allowBackup="false"`.
3. Multiple components exposed directly:

```
<activity android:name="...DoTransfer" android:exported="true"/>
<activity android:name="...ViewStatement" android:exported="true"/>
<activity android:name="...PostLogin" android:exported="true"/>
<activity android:name="...ChangePassword" android:exported="true"/>
<receiver android:name="...MyBroadCastReceiver" android:exported="true"/>
<provider android:name="...TrackUserContentProvider" android:exported="true"/>
```

Any app can directly send Intent to these activities, potentially bypassing authentication and accessing sensitive pages.

The exposed `Provider` allows third parties to read and write the `trackerusers` database.

The exposed `Receiver` can be misused to trigger internal operations (e.g., broadcast injection).

Recommendation: Set `android:exported="false"` or add `android:permission` to restrict access.

5. To Be Continued

- Research on high version Android and APKs in the Android market.
- Study on HTTPS bypass and Frida.
- Further research on ADB usage.