Generative Summary:

This document is a walkthrough for a mobile security training, providing techniques for static and dynamic analysis of a mobile application.

Introduction:

This walkthrough provides an overview of static and dynamic analysis techniques for testing mobile app security. The InjuredAndroid CTF APK can be found at the given GitHub link.

Preparation:

This section provides instructions on the installation and setup requirements for the Android application testing environment.

* Rooted device or emulator running Android 8 is required for the setup.
* The injuredandroid and DrozerAgent APKs need to be installed via adb.
* jadx-gui is recommended for examining the source code.
* FridaServer, apktool, and drozer also need to be installed.
* The apktool command can be used to decompile the injuredandroid APK.
* The drozer console can be started on the device after setting up the adb forward.
* Frida can be used for dynamic analysis, with the frida-server being pushed to the device.
* The frida-ps command can be used to list all processes on the mobile device.

Challenge 1:

This section explains how to decompile an APK using JADX and trigger activities in an Android application. It also provides instructions on finding the package name and listing activities using ADB.

* The section discusses using JADX to decompile an APK and reading the source code of FlagOneLoginActivity.
* It explains how to find the package name and list activities using ADB commands.
* The section mentions the limited list of activities obtained and suggests looking into the AndroidManifest.xml file for exported activities.
* It highlights the b25lActivity as the most likely candidate for triggering an activity.
* The section also mentions an alternative method using Drozer to run activities and trigger the FlagTwo Activity screen.





Challenge 3:

The flag "thr33\_t1m3s" is stored as a string resource with the name "cmVzb3VyY2VzX3lv".



Challenge 4:

The FlagFourActivity compares a decoded Base64 encoded string to the flag data returned by the Encoder.getData method. The data is base64 encoded, not encrypted. By intercepting the getData method using Frida, the flag can be revealed. The process involves overloading the method and converting the byte array to a string.





frida -H 127.0.0.1 -f $(adb shell pm list packages | grep injured | cut - d: -f2) -l flag4decoder.js



Challenge 5:

This section explains how to use Drozer and JADX to list and analyze broadcast receivers in an Android app. It also discusses the use of adb and Frida for debugging and tracing the activation of receivers.

* The app "b3nac.injuredandroid" has a broadcast receiver called "FlagFiveReceiver" with no real parameters.
* Drozer can be used to list the broadcast receivers, but due to a bug, intents are not listed.
* A workaround is to use adb to trigger the receiver using the command "adb shell am broadcast -n b3nac.injuredandroid/.FlagFiveReceiver".
* The code of the receiver can be analyzed using JADX, and a Frida script can be used for debugging and tracing the activation of the receiver.
* When the receiver is triggered, it receives a broadcast with an intent containing the action "Pwned" and extras with the key "s" and value "foo". The flag is obtained after triggering the receiver twice.



adb shell am broadcast -n b3nac.injuredandroid/.FlagFiveReceiver

For debugging I used a simple Frida script to trace the activation of the receiver:

function intercept() {

// Check if frida has located the JNI

if (Java.available) {

// Switch to the Java context

Java.perform(function() {

const myreceiver =

Java.use('b3nac.injuredandroid.FlagFiveReceiver');

myreceiver.onReceive.overload('android.content.Context',

'android.content.Intent').implementation = function (context, intent) {

console.log('[+] received a broadcast');

this.onReceive( context, intent );

}

console.log('[+] FlagFiveReceiver.onReceive hooked')

}

)}

}

walkthrough.md 2/13/2020

21 / 33

intercept()

More elaborate tracing:

function intercept() {

// Check if frida has located the JNI

if (Java.available) {

// Switch to the Java context

Java.perform(function() {

const myreceiver =

Java.use('b3nac.injuredandroid.FlagFiveReceiver');

var Activity = Java.use("android.app.Activity");

var Intent = Java.use("android.content.Intent");

myreceiver.onReceive.overload('android.content.Context',

'android.content.Intent').implementation = function (context, intent) {

console.log('[+] received a broadcast');

var myintent = Java.cast(intent, Intent);

console.log('[+] intent is ' + myintent.toUri(0));

var myaction = myintent.getAction();

var mycomponent = myintent.getComponent();

var myextras = myintent.getExtras();

console.log('[+] action is ' + myaction.toString());

console.log('[+] component is ' + mycomponent.toString());

if( myextras ) {

console.log('[+] extras is ' + myextras.toString());

}

this.onReceive( context, intent );

}

console.log('[+] FlagFiveReceiver.onReceive hooked')

}

)}

}

intercept()

Gives the following output when a message is received:

[+] FlagFiveReceiver.onReceive hooked

[Remote::b3nac.injuredandroid]-> [+] received a broadcast

[+] intent is

#Intent;action=Pwned;launchFlags=0x400010;package=bar;component=b3nac.inju

redandroid/.FlagFiveReceiver;S.s=foo;end

[+] action is Pwned

[+] component is

ComponentInfo{b3nac.injuredandroid/b3nac.injuredandroid.FlagFiveReceiver}

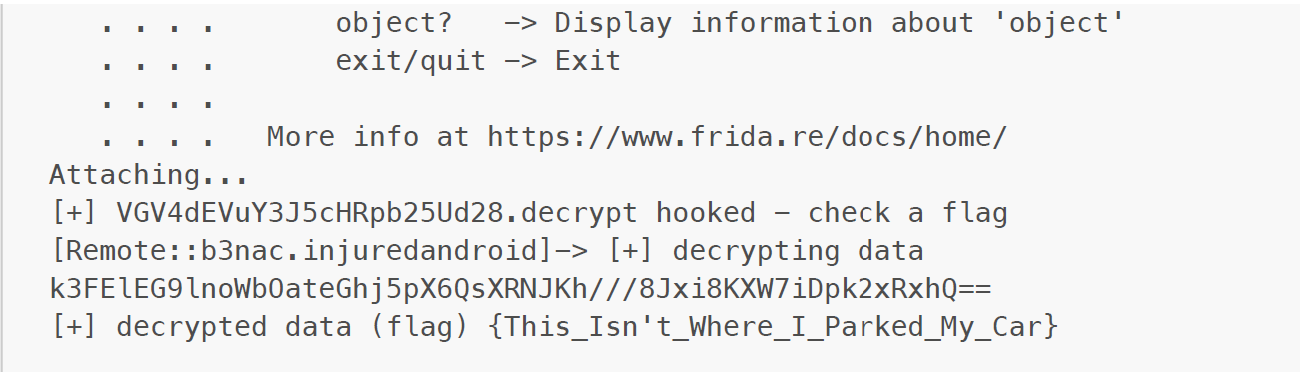
[+] extras is Bundle[{s=foo}]

Challenge 6:

The challenge involves decrypting a flag using Frida and DES encryption. By intercepting the decryption function, the flag is revealed as "{This\_Isn't\_Where\_I\_Parked\_My\_Car}". The process involves using base64 encoded data and the key to decrypt the flag.







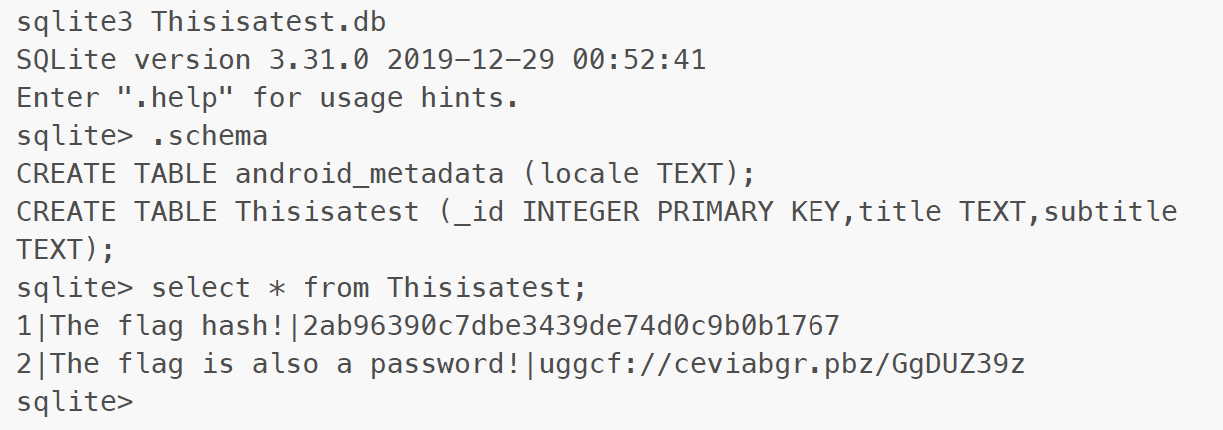
Challenge 7:

This section discusses the SQLite challenge, which involves creating a database, storing data, and solving various tasks related to the challenge.

* The challenge requires completing six previous tasks before starting.
* The code contains three Base64 encoded strings, including a flag hash and a password.
* The remoteUrl is ROT13 encoded and leads to a non-existing note.
* The database file "Thisisatest.db" is located in the data directory of the application.
* The database can be copied to a local system using adb.
* The flag hash "2ab96390c7dbe3439de74d0c9b0b1767" is the MD5 hash for "hunter2".



adb pull /data/data/b3nac.injuredandroid/databases/Thisisatest.db



Challenge 8:

This is about AWS credentials stored in the app, but I could not find any.