



How I Forced An Android Vulnerability Into Bypassing MDM Restrictions + DIY Android Malware Analysis

By

Zubair Ashraf

Presented At

The Digital Forensic Research Conference

DFRWS 2015 EU Dublin, Ireland (Mar 23rd- 26th)

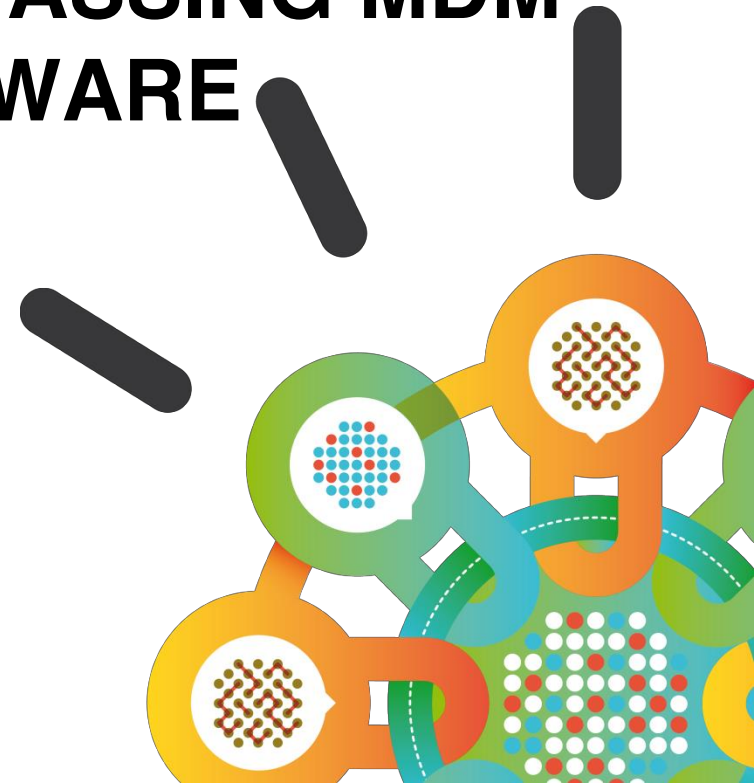
DFRWS is dedicated to the sharing of knowledge and ideas about digital forensics research. Ever since it organized the first open workshop devoted to digital forensics in 2001, DFRWS continues to bring academics and practitioners together in an informal environment. As a non-profit, volunteer organization, DFRWS sponsors technical working groups, annual conferences and challenges to help drive the direction of research and development.

<http://dfrws.org>

Security Intelligence.
Think Integrated.

HOW I FORCED AN ANDROID VULNERABILITY INTO BYPASSING MDM RESTRICTIONS + DIY MALWARE ANALYSIS

Zubair Ashraf
Team Lead & Security Researcher
IBM X-Force Advanced Research



@b0ut.m3

- Team Lead & Security Researcher
@ IBM X-Force Research

 @zashraf1337

 securityintelligence.com/author/zubair-ashraf

 ca.linkedin.com/in/zubairashraf



Agenda

- DIY Malware Analysis (available on slides only 😊)
- Vulnerability Hunt
- Exploitation



Android Malware Analysis

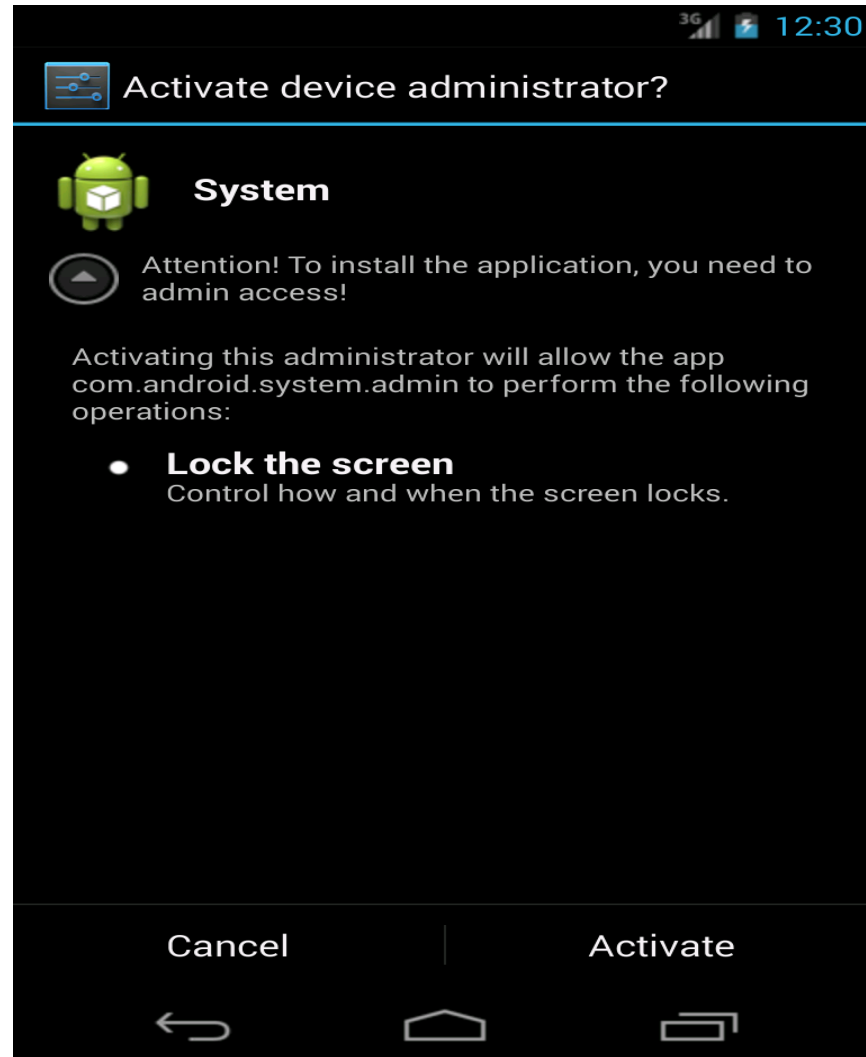
(please refer to slides from download section)

Let's get OBAD in the emulator

```
mobisec@Mobisec:/opt/mobisec/devtools/android-  
sdk/tools$ emulator-arm -avd Android_4.0.3  
-scale 0.75 -debug all -logcat all -no-boot-  
anim
```

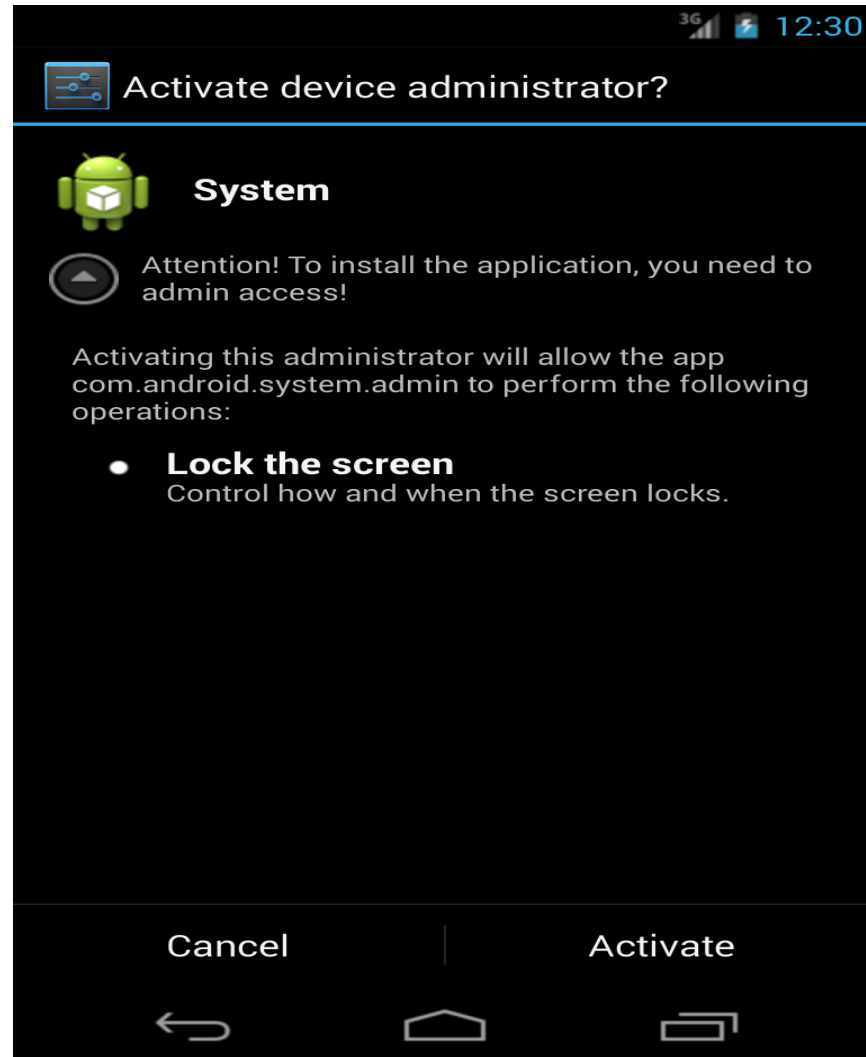
```
mobisec@Mobisec-VM:~$ adb install  
Malware/OBad/E1064BFD836E4C895B569B2DE470028  
4.apk
```

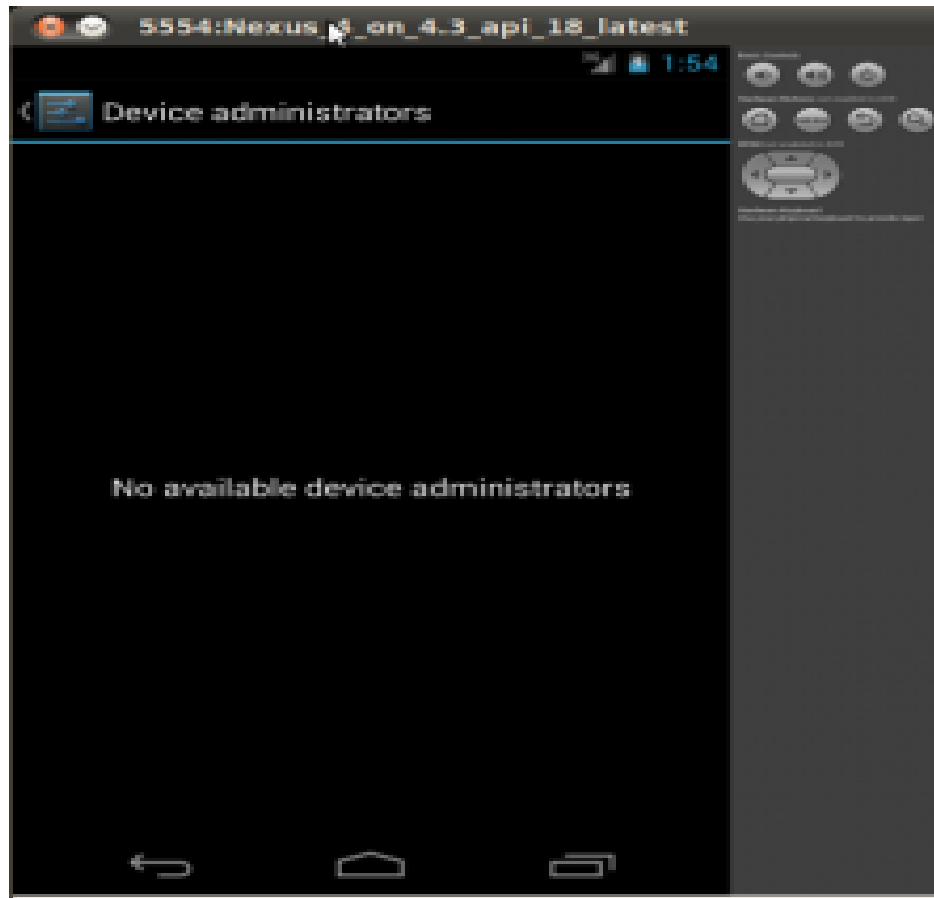
The persistent begging starts



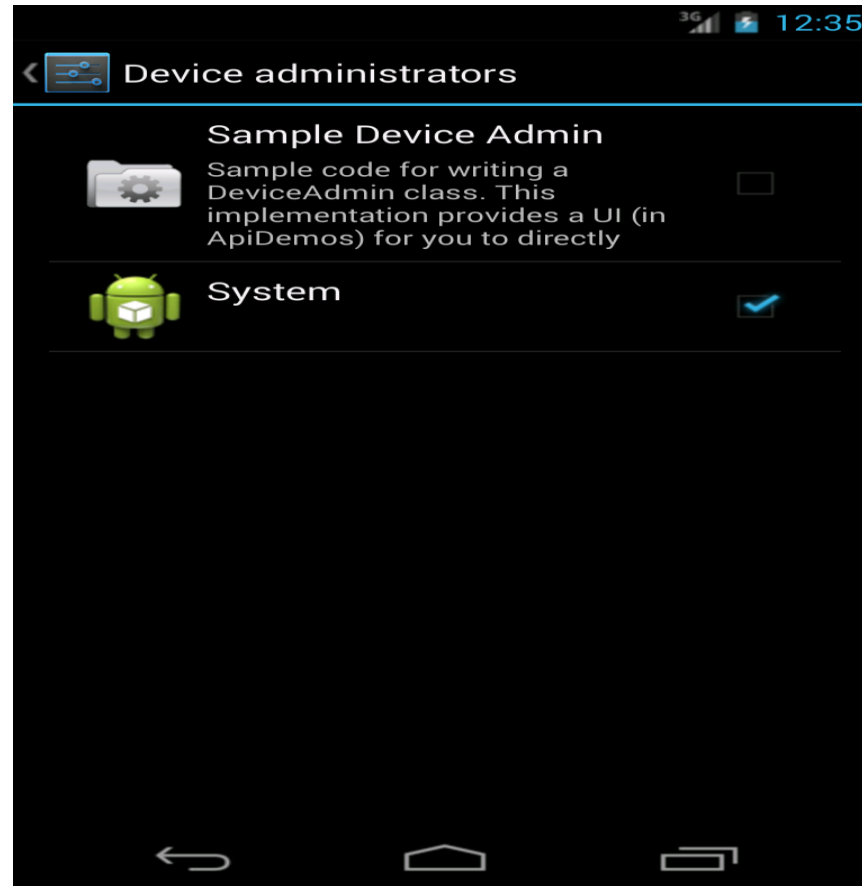


Won't take No for an answer

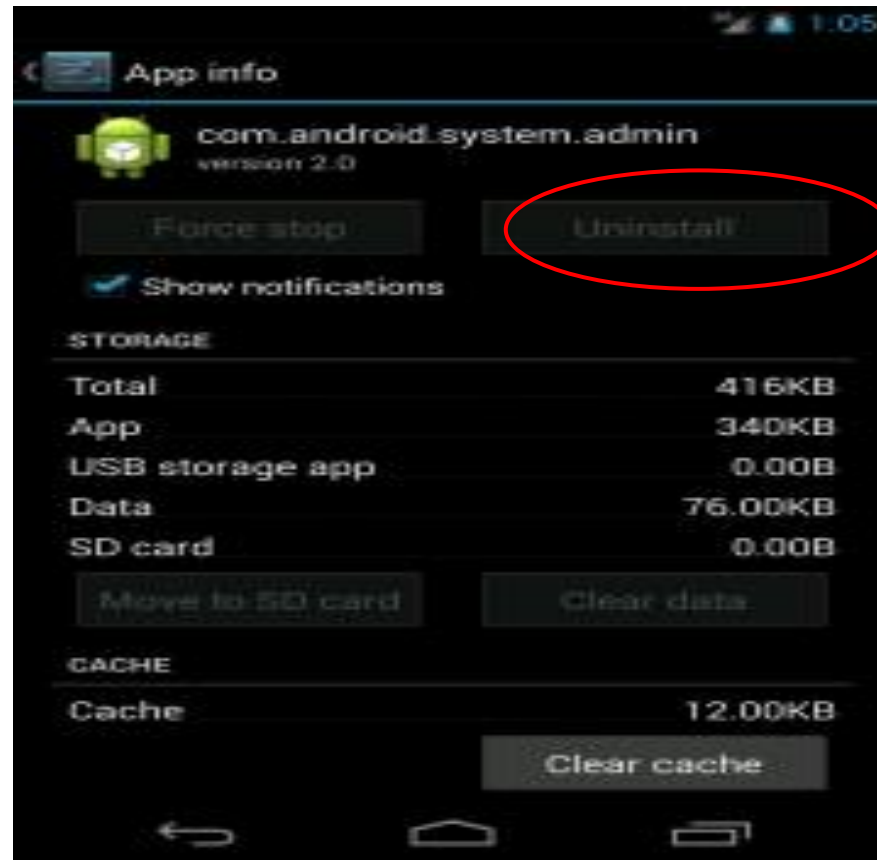




No Device Admin?



We would expect something like this



Can we see OBAD in app list and uninstall it?

Let's try the command line

- mobisec@Mobisec-VM:~/Malware/OBAD\$ adb
uninstall com.android.system.admin
Failure

```
mobisec@Mobisec-VM:~/Malware/OBAD$ adb  
logcat -d -b main -b events | grep admin | tail -1
```

```
W/PackageManager( 277): Not removing  
package com.android.system.admin: has  
active device admin
```

May be from command line - 'adb'

Let's hunt the code that hides it from Device Admin List

Checkout the patch history ... or ...

Find Relevant Code

Launch *Settings* -> *Security* -> *Device Administrators*

Check out the logs:

```
adb logcat -d -b events
```

```
I/am_new_intent( 276):
```

```
[0,1106566944,17,com.android.settings/.Settings,android.intent.action.  
MAIN,NULL,NULL,274726912]
```

```
I/am_resume_activity( 276):
```

```
[0,1106900904,17,com.android.settings/.Settings]
```

```
I/am_on_resume_called( 1118): [0,com.android.settings.Settings]
```

Find Relevant Code (contd...)

- search for these strings at
`androidxref.com`
- following along you will arrive at
`packages/apps/Settings/src/com/android/settings/
DeviceAdminSettings.java`

Find Relevant Code (contd...)

- check out the function

void updateList()

- and the conditions for something to appear in device admin list

Device Admin Vulnerability

```
getActivity().getPackageManager().queryBroadcastReceivers(Intent(DeviceAdminReceiver.ACTION_DEVICE_ADMIN_ENABLED), ...
```

Device Admin Vulnerability

```
getActivity().getPackageManager().queryBroadcastReceivers(Intent(DeviceAdminReceiver.ACTION_DEVICE_ADMIN_ENABLED), ...
```

Hackers won't follow the specs unless they have to

What they should do

To use the Device Administration API, the application's manifest must include the following:

- A subclass of [DeviceAdminReceiver](#) that includes the following:
 - The [BIND_DEVICE_ADMIN](#) permission.
 - The ability to respond to the [ACTION_DEVICE_ADMIN_ENABLED](#) intent, expressed in the manifest as an intent filter.

What they actually did

```
<receiver "System"=".OCllCoO">  
  <meta-data "android.app.device_admin"  
="@2130968576">  
  </meta-data>  
  <intent-filter>  
    <action  
name="com.strain.admin.DEVICE_ADMIN_ENABLED">  
    </action>  
  </intent-filter>  
</receiver>
```

What they actually did

instead of

android.app.action.DEVICE_ADMIN_ENABLED

name="com.**strain**.admin.DEVICE_ADMIN_ENABLED">

What's next

Device Admin Vulnerability

`services/java/com/android/server/`

`DevicePolicyManagerService.java`

Device Admin Vulnerability

When adding an Admin

```
policy.mAdminMap.put(adminReceiver, newAdmin);
```

and

```
policy.mAdminList.add(newAdmin);
```

Device Admin Vulnerability


Please make sure you take ALL your stuff with you

Device Admin Vulnerability

removeActiveAdminLocked

- 1.policy.mAdminList.remove(admin);
- 2.policy.mAdminMap.remove(adminReceiver);

Device Admin Vulnerability



ALL THE TIME! even when in
RUSH

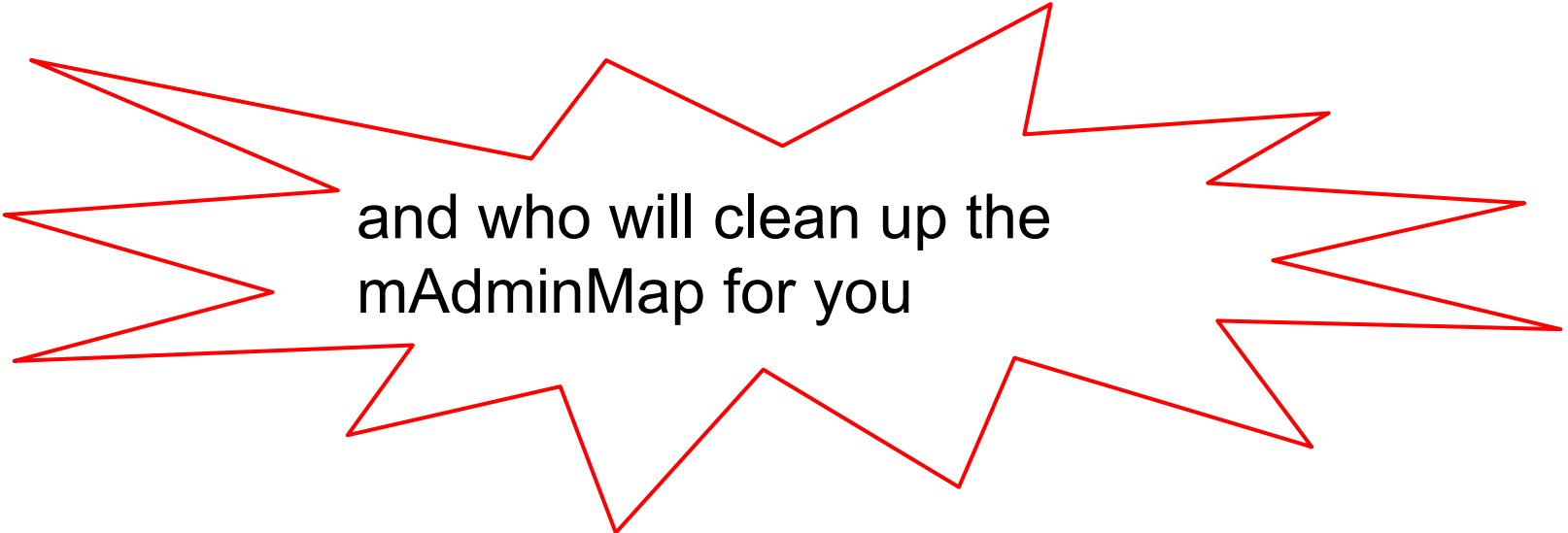
Please make sure you take ALL your stuff
with you

Device Admin Vulnerability

```
private void handlePackagesChanged(int userHandle) {  
  
    removed = true;  
    policy.mAdminList.remove(i);  
}
```

Device Admin Vulnerability

```
private void handlePackagesChanged(int userHandle) {  
  
    removed = true;  
    policy.mAdminList.remove(i);  
}
```



and who will clean up the
mAdminMap for you

Device Admin Vulnerability

This code path gets executed when you DISABLE the device admin component

Device Admin Vulnerability

All we have so far is a leak / bad coding practice

Device Admin Vulnerability

Is this a vulnerability?

Device Admin Vulnerability

Is there a code path that consults mAdminMap but not mAdminList ?

Device Admin Vulnerability

- `getActiveAdminUncheckedLocked`
- `getActiveAdminForCallerLocked`
(ComponentName who, int reqPolicy)
with “who” parameter being non null

Device Admin Vulnerability

getActiveAdminUncheckedLocked is used by [isAdminActive](#)

So can we exploit it?



MDM

Gartner

DID YOU KNOW? By 2016, 20% of enterprise BYOD programs will fail due to deployment of **mobile device management (MDM)** measures that are too restrictive.



How about typing a 14 character password while driving?

Exploiting the Device Admin Vulnerability

- enable device admin
- disable the device admin component
- At this point, from the data structure and code perspective, device admin's `isAdminEnabled` will still return true

Exploiting the Device Admin Vulnerability

```
pm.setComponentEnabledSetting(  
    this.getWho(context),
```

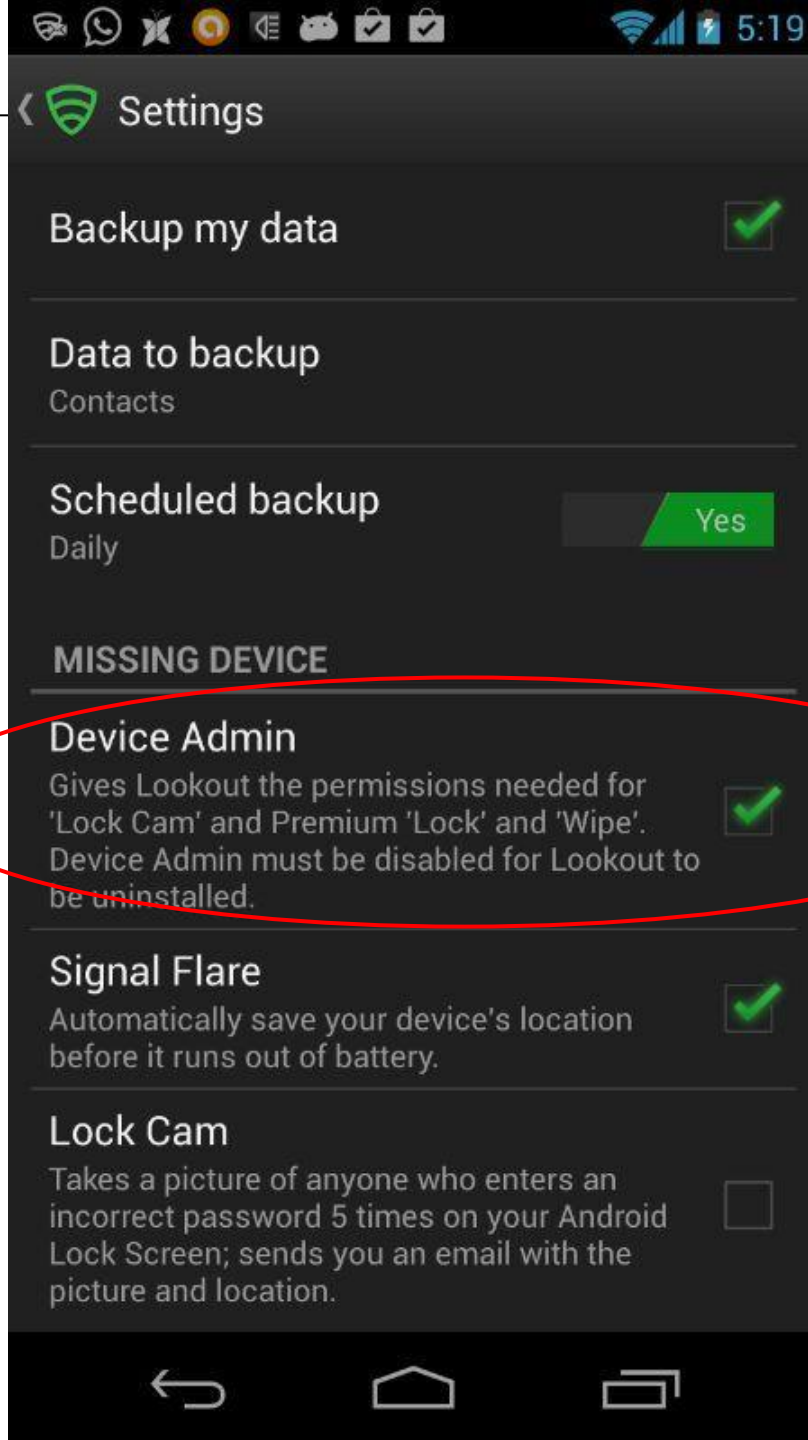
```
PackageManager.COMPONENT_ENABLED_STATE_DISABLED,  
PackageManager.DONT_KILL_APP);
```

Exploiting the Device Admin Vulnerability

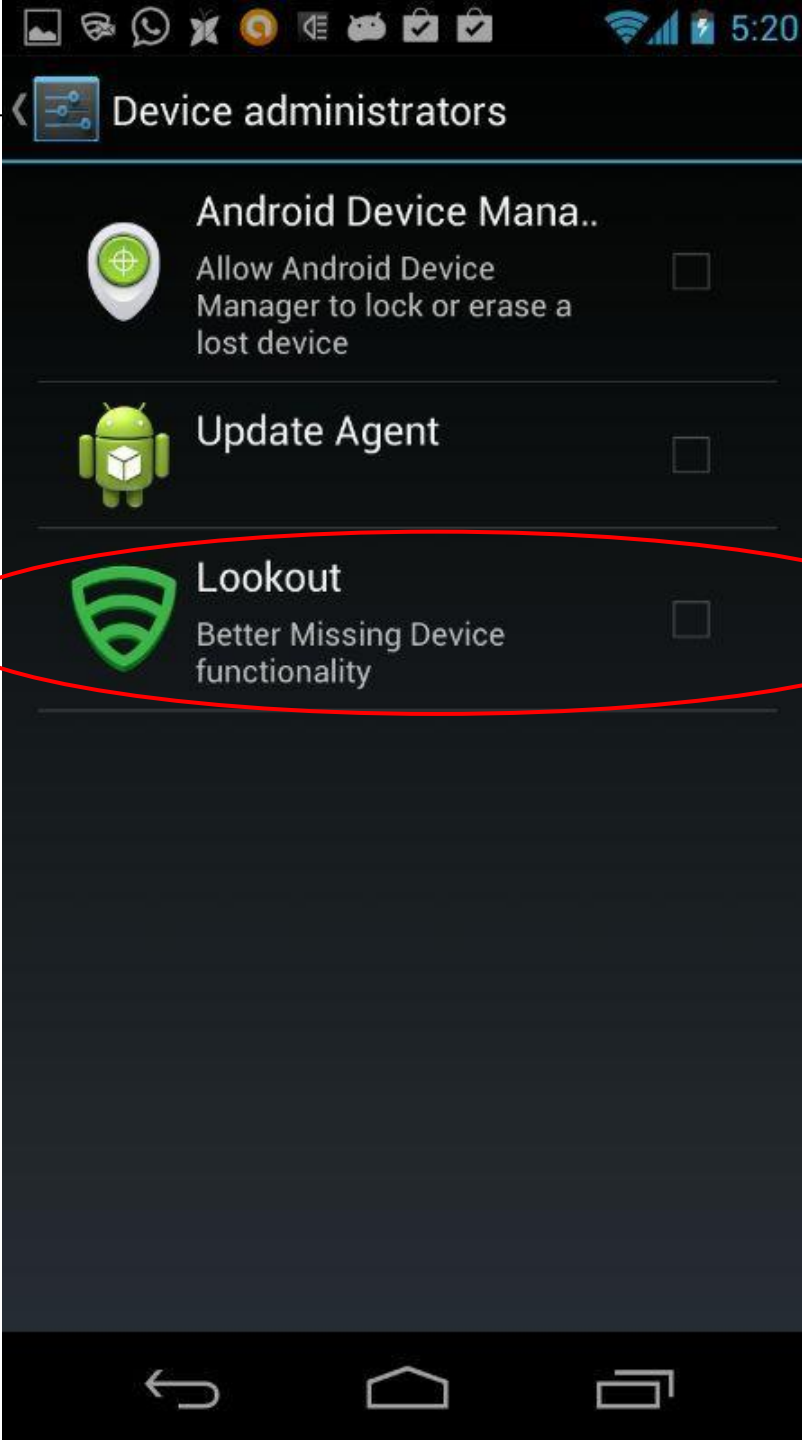
Uninstall the app (it will still be in the mAdminMap)

Exploiting the Device Admin Vulnerability

Now, install the original app



BUT



BUT it may not necessarily work with MDM



isActivePasswordSufficient

isActivePasswordSufficient

```
public boolean isActivePasswordSufficient(int userHandle)
{
    enforceCrossUserPermission(userHandle);
    synchronized (this) {
        // This API can only be called by an active device
admin,
        DevicePolicyData policy = getUserData(userHandle);
        // so try to retrieve it to check that the caller is one.
getActiveAdminForCallerLocked(null,
        DeviceAdminInfo.USES_POLICY_LIMIT_PASSWORD);
```

getActiveAdminForCallerLocked

```
ActiveAdmin getActiveAdminForCallerLocked  
  (ComponentName who, int reqPolicy) throws  
    SecurityException {  
  if (who != null) { ... }  
  else {  
    final int N = policy.mAdminList.size();
```


getActiveAdminForCallerLocked

```
else {  
    final int N = policy.mAdminList.size();  
    for (int i=0; i<N; i++) {  
        ActiveAdmin admin = policy.mAdminList.get(i);  
        if (admin.getUid() == callingUid &&  
            admin.info.usesPolicy(reqPolicy)) {  
            return admin;  
        }  
    }  
    }  
    throw new SecurityException
```

getActiveAdminForCallerLocked

```
else {  
    final int N = policy.mAdminList.size();  
    for (int i=0; i<N; i++) {  
        ActiveAdmin admin = policy.mAdminList.get(i);  
        if (admin.getUid() == callingUid &&  
            admin.info.usesPolicy(reqPolicy)) {  
            return admin;  
        }  
    }  
    throw new SecurityException
```

There is a way

sharedUID

- active device admin with same policies
- and same UID - sharedUID

```
if (admin.getUid() == callingUid &&  
    admin.info.usesPolicy(reqPolicy)) {
```

Extended Hack

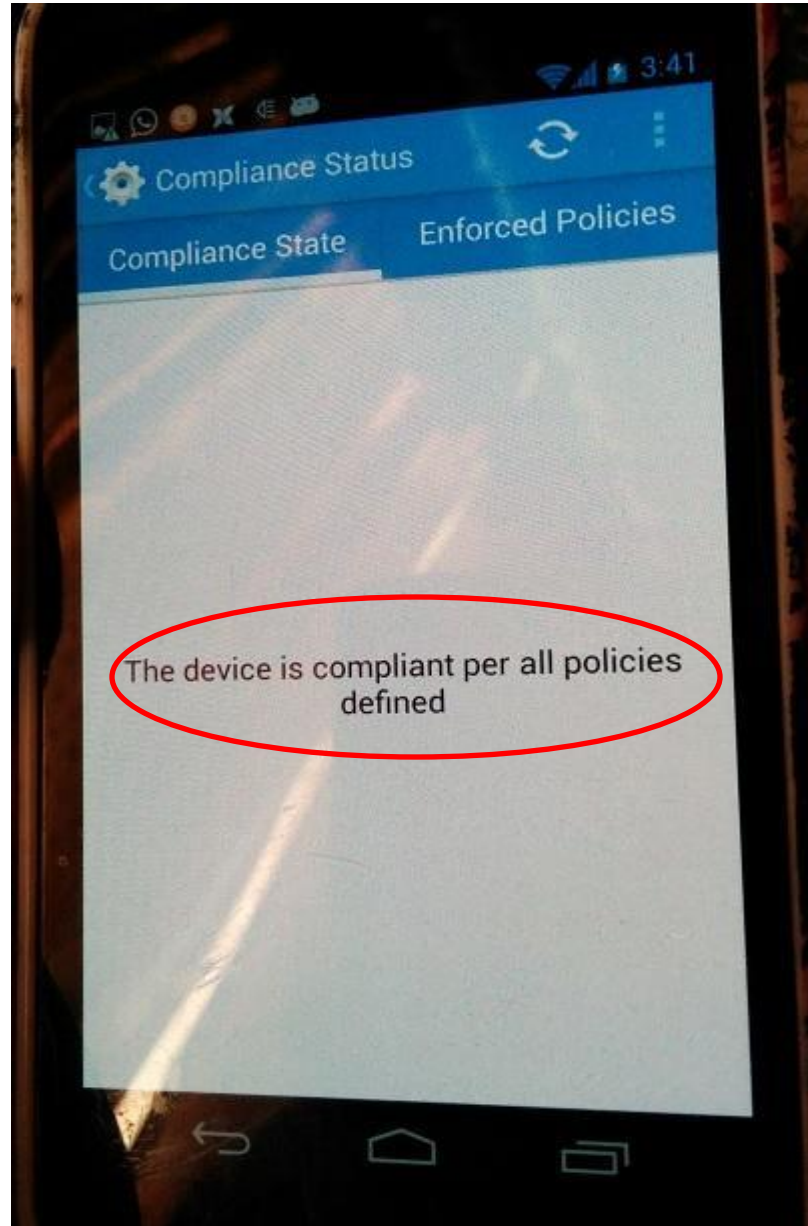
- Modify AndroidManifest.xml of the MDM
 - add android:sharedUserId attribute
- repackage and self sign

Extended Hack

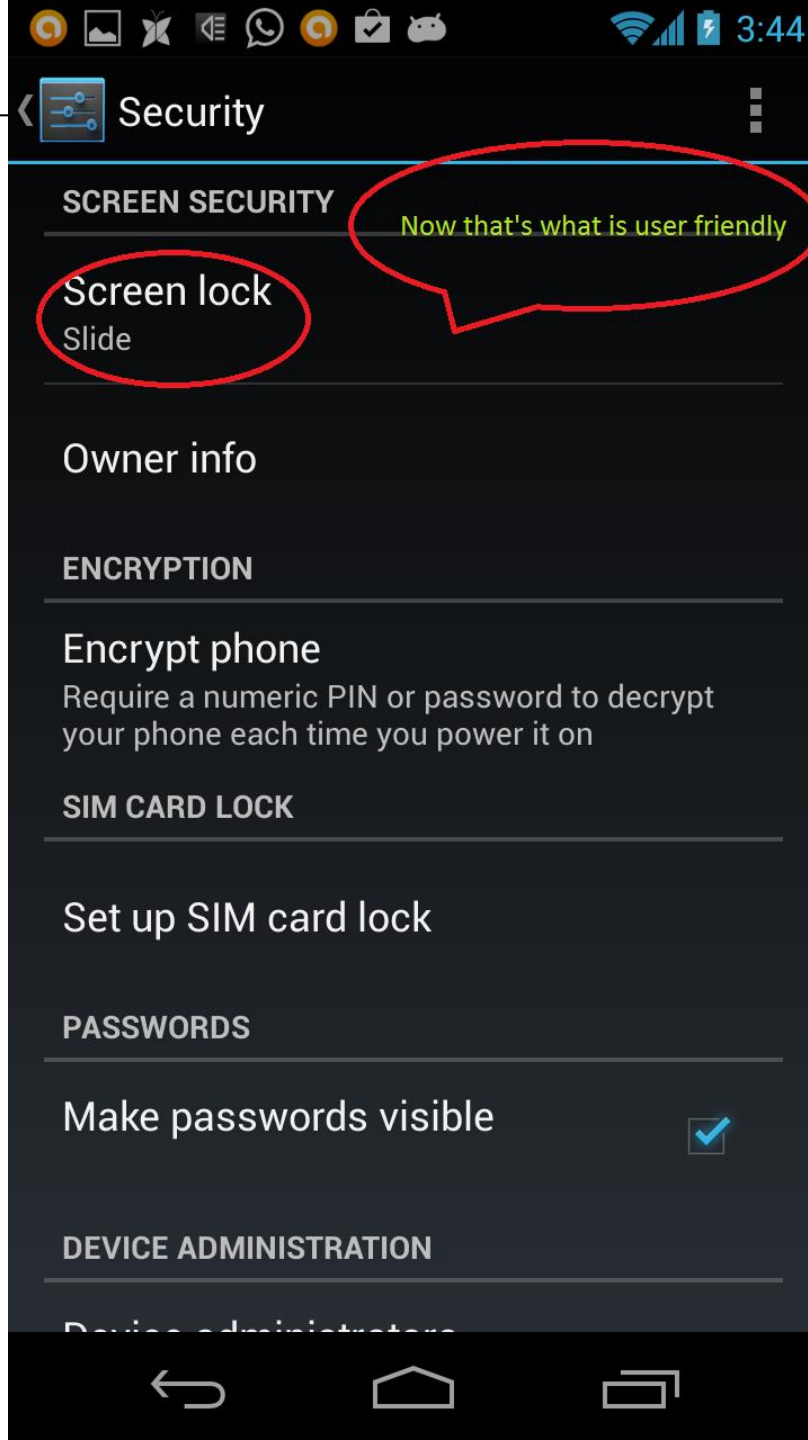
- Create a different device admin
 - same sharedUid
 - same policies
 - install and activate it

Extended Hack

- Do everything else as before
- but using the self signed MDM apk with sharedUID



COMPLIANT != SECURE



Lessons

- Don't make it really painful to use the device
- code protection
- verifying app signatures



Further Learning

- <https://github.com/strazzere/android-unpacker>
- <https://github.com/strazzere/android-unpacker/blob/master/AHPL0.pdf>

Loved ones, X-Force, DFRW EU and YOU



QUESTIONS

 @zashraf1337

 securityintelligence.com/author/zubair-ashraf

 ca.linkedin.com/in/zubairashraf