50 Ways to Leak Your Data: An Exploration of Apps’ Circumvention of the Android Permissions System

Joel Reardon, Álvero Feal, Primal Wijesekera, Amit Elazari Bar On, Narseo Vallina-Rodriguez, Serge Egelman

August 15th, 2019
Apps need **permissions** to access **resources**
App permissions

Kaloer Clock - Alarm Clock needs access to:

In-app purchases
Make purchases within the app

Your messages
Read your text messages (SMS or MMS), receive text messages (SMS)

System tools
Install shortcuts

Your location
Approximate location (network-based), precise location (GPS and network-based)

Your personal information
Read calendar events plus confidential information

Your accounts
Add or remove accounts, use accounts on the device

Lock screen
Despite the failures of permission systems they serve an **important purpose.**
Despite the failures of permission systems they serve an **important purpose**. At the very least, if an app is **denied** permission, it **must not** access resources protected by the permission.
In systems security, however, security mechanism can often be circumvented or avoided.
Two common circumventions are **covert channels** and **side channels**.
Alice app

allow access

covert channel

Security mechanism

Deny access

Bob app
eve app

deny access

security mechanism

side channel
We search for evidence of side or covert channels actually being used, and then figure out what the channels are.
Method
app
corpus

access
to
allowed
data
app
corpus
access
to
allowed
data
reverse engineering
app
corpus
cheat
apps
that
access
to
allowed
sent
out
data data
set minus
!!
ok
alert

app corpus

apps that cheat

reverse engineering

set minus
data sent out
data allowed to access
reverse engineering apps that cheat

set minus

side chan. cov. chan.

data sent out data allowed to access

app corpus

ok alert
SDKs are to blame

- most behaviours were from SDKs included in an app
  - SDKs inherent permissions from apps
- helped reverse engineering
  - only need to reverse one SDK for all apps that use it
- helped measure prevalence
  - we could look for fingerprint of malicious code
apps
<table>
<thead>
<tr>
<th>apps</th>
<th>3rd party</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
dynamic testing

{ apps

3rd party

IMEI

IMEI
Dynamic testing

- Apps
- Malicious code
- IMEI (but has permission)

3rd party
dynamic testing

apps

IMEI

IMEI

IMEI (but has permission)

3rd party
Dynamic testing

Apps

IMEI

IMEI (but has permission)

3rd party

Static testing

Same malicious code but not executed
dynamic bound at least X apps DO
at least $X$ apps DO

apps

dynamic bound

static bound
Dynamic bound

at least X
apps DO

Static bound

at least Y
apps COULD

apps
Highlight One
Highlight One

Chinese company Baidu uses **external storage** as a covert channel.
/sdcard
/backups
/.SystemConfig
/.cuid

........f.....JI5.....e:.W.
........ra6..8F.`...........
........n...f..2~
:cond_0
:try_start_0
const-string v0, "30212102dicudiab"

const-string v1, "30212102dicudiab"

invoke-virtual {p0}, Ljava/lang/String:-˃getBytes()[[B

move-result-object v2

invoke-static {v0, v1, v2}, Lcom/baidu/android/bbalbs/common/a/a:-˃a(Ljava/lang/String:Ljava/lang/String;[B)[B
:cond_0
:try_start_0
const-string v0. "30212102dicudiab"
const-string v1. "30212102dicudiab"

invoke-virtual {p0}.Ljava/lang/String:->getBytes()[B

move-result-object v2

invoke-static {v0, v1, v2}.Lcom/baidu/android/bbalbs/common/a/a:->a(Ljava/lang/String;Ljava/lang/String;[B)[B

<- baiducid20121203
:cond_0
:try_start_0
const-string v0. "30212102dicudiab" <- baiducid20121203
const-string v1. "30212102dicudiab"
invoke-virtual {p0}. Ljava/lang/String:->getBytes()B
move-result-object v2 33303231323130326469637564696162
invoke-static {v0, v1, v2}. Lcom/baidu/android/bbalbs/common/a/a:->a(Ljava/lang/String;Ljava/lang/String;[B)[B
{  
  "device_id":  
     "your_device_id",
  "imeis": "your_imei",
  "version": 2
}

/sdcard
/backups
/.SystemConfig
/.cuid
IMEI to Baidu

153 apps had encryption key
IMEI to Baidu

73 send IMEI
153 apps had encryption key
IMEI to Baidu

- 153 apps had encryption key
- 20 didn’t have IMEI permission
- 153 apps had encryption key
- 20 didn’t have IMEI permission
IMEI to Baidu

8 actually sent IMEI
20 didn’t have IMEI permission

153 apps had encryption key
IMEI to Baidu

153 apps had encryption key
8 actually sent IMEI
20 didn’t have IMEI permission
153 apps had encryption key
153 apps had encryption key
8 actually sent IMEI
one made by Disney
20 didn’t have IMEI permission
153 apps had encryption key
IMEI to Baidu

700 million installs

- 8 actually sent IMEI
- 20 didn’t have IMEI permission
- one made by Disney

153 apps had encryption key
Highlight Two
American company Unity uses *ioctls* as a *side channel*
American company Unity uses **ioctl**s as a **side channel** to get phone’s MAC address through a **SIOCIFHWADDR ioctl**.
MAC Address to Unity

711 sent MAC
MAC Address to Unity

42 sent MAC without permission

711 sent MAC
MAC Address to Unity

42 sent MAC without permission

711 sent MAC

12408 have ioctl code
MAC Address to Unity

42 sent MAC without permission

711 sent MAC

2 billion installs

12408 have ioctl code
Other findings include **geolocation** from EXIF metadata; getting the **router MAC address** through UPnP or the OS’s ARP cache (/proc/net/arp).
Limitations and Future Work

- we focused on a subset of permissions
- we needed *conspicuous* network transmissions
- we rely on the Android Exerciser Monkey as a UI fuzzer
- we only studied apps from the Google Play Store
OpenX

OpenX is an advertising company that we found collecting the MAC addresses of the WiFi router, which can be used as a surrogate for location. It was able to do this without any permissions by reading the ARP cache on the device. The following apps were sending the router information without holding the ACCESS_WIFI_STATE permission:

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Version</th>
<th>MD5 hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.digitalchemy.calculator.freedecimal</td>
<td>5510</td>
<td>2773b4c584b9663558aab7c175f661e9</td>
</tr>
<tr>
<td>com.digitalchemy.calculator.freedecimal</td>
<td>5620</td>
<td>747464fb190d0913ed38760c50e6ca99</td>
</tr>
<tr>
<td>com.digitalchemy.calculator.freedecimal</td>
<td>5660</td>
<td>08e910230670e0523ae4d2e78a2f7aad</td>
</tr>
<tr>
<td>com.digitalchemy.calculator.freedecimal</td>
<td>5710</td>
<td>e99d91fcd965de051305cfb443237a6</td>
</tr>
<tr>
<td>com.mobilityware.PyramidFree</td>
<td>2318</td>
<td>f5c829c56e111a4a0d5e5b691f888049</td>
</tr>
</tbody>
</table>
Apps using Side and Covert Channels

Posted by Joel Reardon on June 1, 2019

The 2019 USENIX Security Symposium includes our study on the use of side and covert channels by apps in order to circumvent the permission system. In addition to our paper, we have a series of blog posts to explain our main findings. In this post we provide the name, version, and MD5 hash of the... Continue reading →

Why do you even need the IMEI?

Posted by Joel Reardon on April 26, 2019

The International Mobile Equipment Identity, or IMEI, is a special number that is tied to every unique mobile phone. It is used whenever the phone is registered on a mobile network. It can also be used to blacklist a phone off of the network, which happens, for example, when it is reported as stolen. Blacklisting... Continue reading →

Ad IDs Behaving Badly

Posted by Serge Egelman on February 14, 2019
we search for **evidence** of side and covert channel **in use**
- we found **exploited** bugs and storage covert channel **use**
- SDK typically responsible
  - how to trust?
**is the permission system **notice and consent**?**
- we received a bug bounty for our work
  - commitment to fix in Q
  - privacy as a luxury?
  - why not over the air?
- disclosure at: blog.appcensus.mobi