As smartphones continue to close the hardware gap with personal computers and laptops, users are running an increasing number of applications to achieve the most out of their phone’s resources. With the Android operating system being the leader of market shares, there exists a need for sound security. Android has many security flaws that are being exploited by malicious developers on a regular basis. The features that have been put in place by the Android team have proven to be insufficient in preventing malware from proliferating through official and unofficial repositories. Private data leakage has become a popular topic because of the sheer number of applications that request the permissions to access the private data and an influx of general privacy concerns amongst the Android community.

Explicit Data Flows — secret variables are explicitly leaked to publicly observed variables.
PRIVATE VAR secure
PUBLIC VAR notsecure
notsecure = secure

Implicit Data Flows — secret variables are leaked through program control flow
PRIVATE VAR secure
PUBLIC VAR notsecure
if secure="123" then:
insecure=1
Else:
insecure=0

TaintDroid Evasion Techniques
(1) Convert tainted variable to ASCII array
(2) Loop array while comparing each element with known ASCII value.
(3) Store ASCII value in a new array, untainted.
(4) Convert value into string to pass outside of the system.

Bitmap Pixel Attack
(1) Set an image pixel to the value of the tainted variable.
(2) Access the rendered image to retrieve the untainted variable.

A PC equipped with an Intel (i7) 3.4 GHZ, 16 GB of DDR3 RAM, and 1.5 TB hard disk on Windows 7 was used for research purposes. The analysis took place in a virtualized environment using Oracle VirtualBox and Ubuntu 14.04 64 bit operating system. Applications were analyzed using Droidbox, which is currently without documentation but available source code indicates TaintDroid is used for taint analysis. Android SDK+ADT tools were downloaded from official sources and used to run a Nexus 4 emulator on Android 4.1.2 using ARM for the CPU. Droidbox outputs JSON logs for each application that contains labeled actions performed while analysis occurred. Any “dataleak” label contained in the logs symbolized that a leak occurred during the analysis. Applications that were flagged for a data leak are then put under another analysis using APKInspektor to view the Android manifest file and a list of the permissions requested.

Example of JSON Log
{
    'DataLeak': { // data leak via SMS
        "sink": "SMS",
        "number": "0735005281",
        "tag": "0x400",
        "data": "dbd4e36bd5295531800e9596724361c4"
    }
}