Policy enforcement within TaintDroid (CSE-509 Class Project)

Anchal Agarwal (108997912)
Aakrati Talati (108996980)
Abhiroop Dabral (108200700)

OUTLINE

What is TaintDroid?

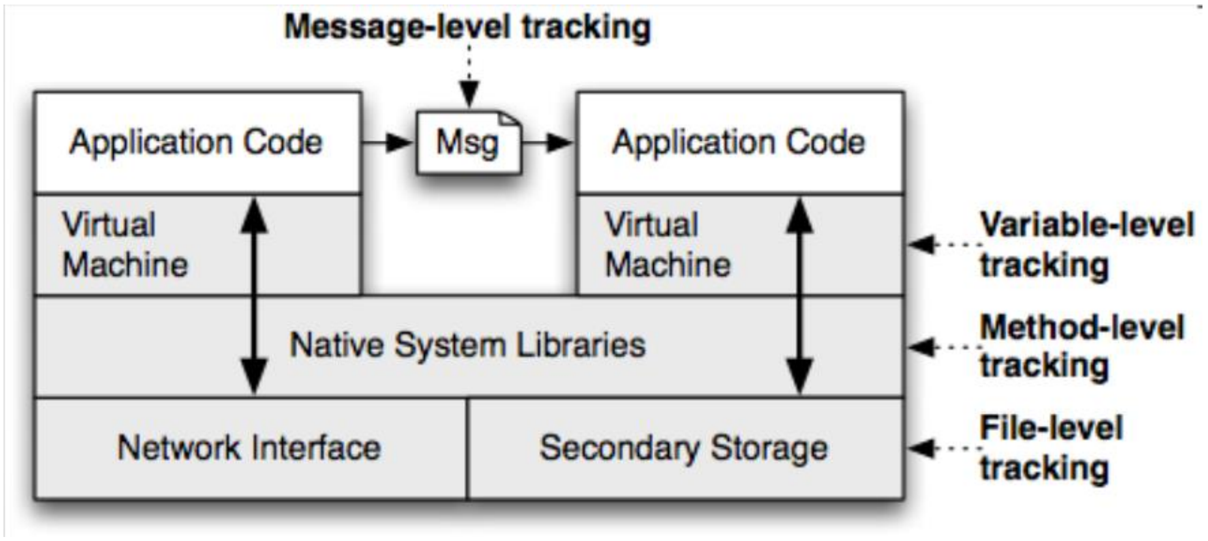
- TaintDroid is software developed for Android that tracks information flow in Android applications.
- TaintDroid is an example of a dynamic analysis system of IF.
- The source code of TaintDroid is available at: www.appanalysis.org
- TaintDroid provides realtime analysis by leveraging Android's virtualized execution environment.
- Monitoring sensitive data with TaintDroid provides informed use of third-party applications for phone users and valuable input for smart- phone security service firms seeking to identify misbehaving applications.
- The primary goals are to detect when sensitive data leaves the system via untrusted applications and to facilitate analysis of applications by phone users or external security services.

Why TaintDroid?

- Applications on Android Market are not verified by Google (which is the case in AppStore)
- Developers can only request coarse-grained permissions through Manifests.
- Easy control of how private information flows, instead of reading not so understandable Manifests (Android), or getting unnecessary prompts each time such information is accessed (iOS).
- Users rarely reads or understands the meaning of the permissions

Design of TaintDroid

Levels of Taint Tracking with TaintDroid



TaintDroid taint sources

- GPS
- Files on SD-card
- Contacts
- Accelerometer
- Microphone
- Camera
- SMS
- Sim card data
- IMEI Number

TaintDroid taint sinks

- WiFi
- 3G
- Bluetooth
- SMS
- NFC

Design Approach

- Monitored taint logs to identify the proper places for policy enforcement.

- ***Declassifier*** class is created to provide an unforgeable capability to the application which can be used to set the NO_TAINT flag on data object that application wants to send on Network***.***

- ***Declassifier*** class is a Singleton ***public*** class. Since the trusted UI is not implemented, the class is kept public for demo purpose. The purpose of keeping class Singleton is that single instance of declassifier is instantiated and this object must be available throughout the application without the need of explicitly passing to each and every method. So that whenever application needs it can call getInstance() on Declassifier and untaint the data.

- ***Declassifier*** class has defined source and sink class members, however when a secure UI is implemented the source and sink values should be set by the OS itself. In future, based on these Sink and Source flags you can selectively declassify data.

- OnClick() of a button, a flag is set to true in Declassifier class which states that the data has to be declassified. However, once the secure UI is implemented the object along with the declassifier object should be instantiated by the OS.

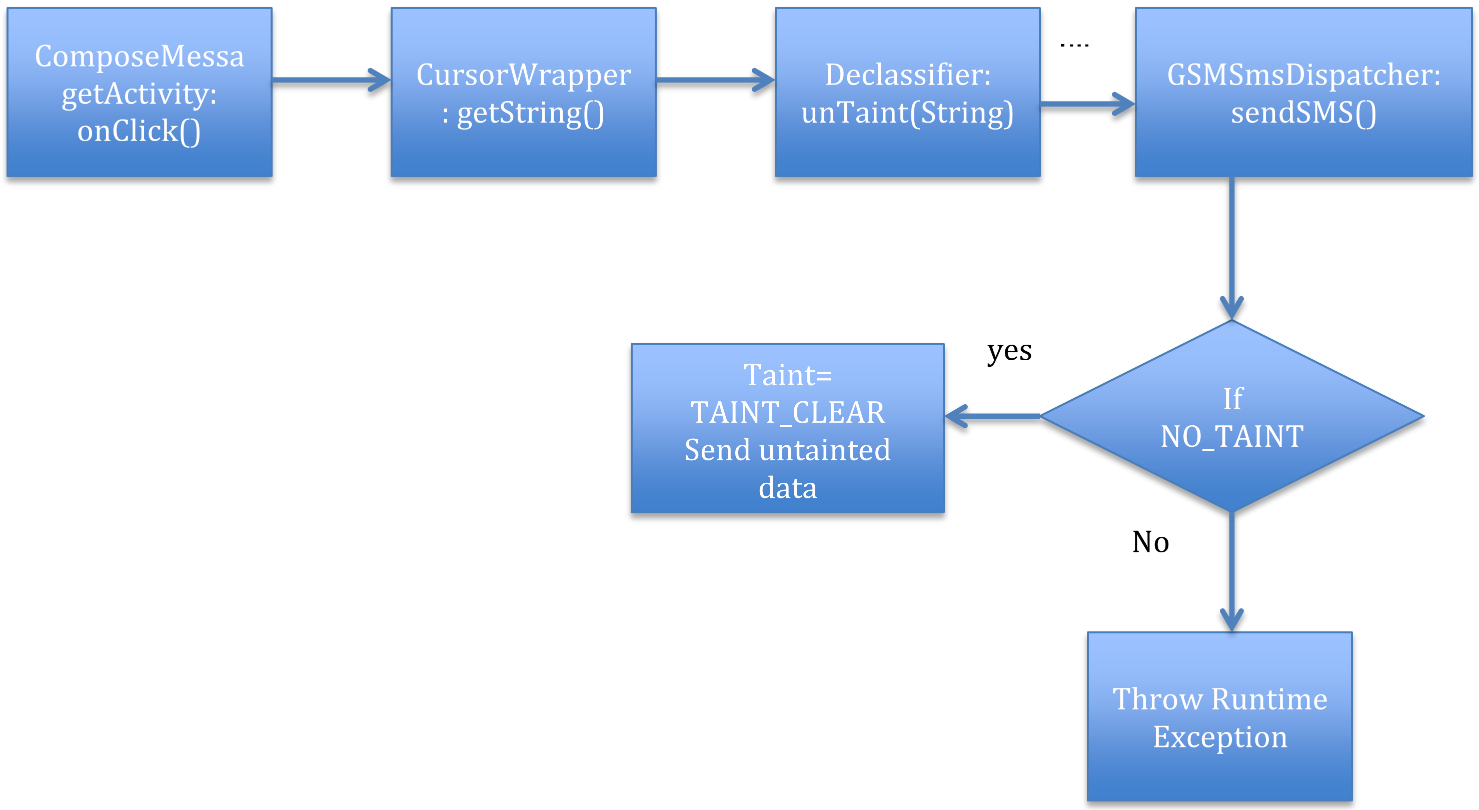
- ***Declassifier*** will set the NO_TAINT flag for the taint byte array for string object passed.

- Added our own taint flag ***NO_TAINT*** in Taint.java, to add flexibility to the working of TaintDroid and simply not cleared all the existing taint tags. At sink site, data will have all the taint bits set by TaintDroid in the taint byte array and also our flag NO_TAINT.

Policy Enforcement within TaintDroid

- Identified sink points and if the data is tainted at that point Run time exception is thrown and TaintDroid displays the notification.
- OnClick() of secure button(assumed) instantiates Declassifer and sets the flag to true and based on this flag further we set our own NO_TAINT bit to the data.
- Added a new taint bit flag "***NO_TAINT***" to Taint.java. The bit is set depending upon whether we want to classify the data or not. The taint bit can be set for all data types, however, we have just set it for String values in CursorWrapper.java
- We have modified existing android messaging application to test our policy.

Eg: Control Flow for SMS application


ComposeMessa getActivity: onClick()
CursorWrapper : getString()
Declassifier: unTaint(String)
....
GSMSmsDispatcher: sendSMS()
If NO_TAINT
yes
Taint= TAINT_CLEAR Send untainted data
No
Throw Runtime Exception

- The *benign* application will not send the sensitive information to a sink without explicit trigger from the user (which can be pressing of a button.)

- Whenever user presses the Send button on the SMS application, which we ***assume*** to be a secure button, flag on declassifier object is set. Data will then be declassified i.e untainted and taint notification will *NOT* be seen. We can send untainted data over the network.

- On the other hand if a malicious app tries to send the sensitive information without the actual consent of the user the *NO_TAINT* flag is not set and taint notification will be shown and a Runtime Exception is thrown.

Source Code Structure

Package	**Source File Name**	**What Changed?**
Libcore/dalvik/src/main/java (dalvik.system)	Declassifier.java	New Declassifier class created to set the declassifier flag on data. If declassifier flag =true, untaint method adds NO_TAINT bit on string object passed to it.
Libcore/dalvik/src/main/java (dalvik.system)	Taint.java	NO_TAINT flag bit added to use one of the unused bits (0x0010000) on taint byte array associated with all objects in TaintDroid. This can be further extended to emulate the functionality of TaintDroid and can introduce multiple taint flag bits for each SINKS. Eg NO_TAINT_SMS, NO_TAINT_GPS.
Package/apps/Mms/src (com.android.mms.ui)	ComposeMessageActivity.java	Modified onClick() to get instance of Declassifier which will be further used for untainting Data.
Framework/base/core/java (android.database)	CursorWrapper.java	SMS application while sending Sms data calls getString() method of this class, we modified it to call Declassifier's untaint method.
Frameworks/base/telephony/java (com.android.internal.telephony.gsm)	GsmSMSDispatcher.java	Modified sendSMS method to check if our NO_TAINT flag is set on the taint byte array of pdu object. If it is that means data is meant to be declassified else if any other flag is set and data was not declassified a

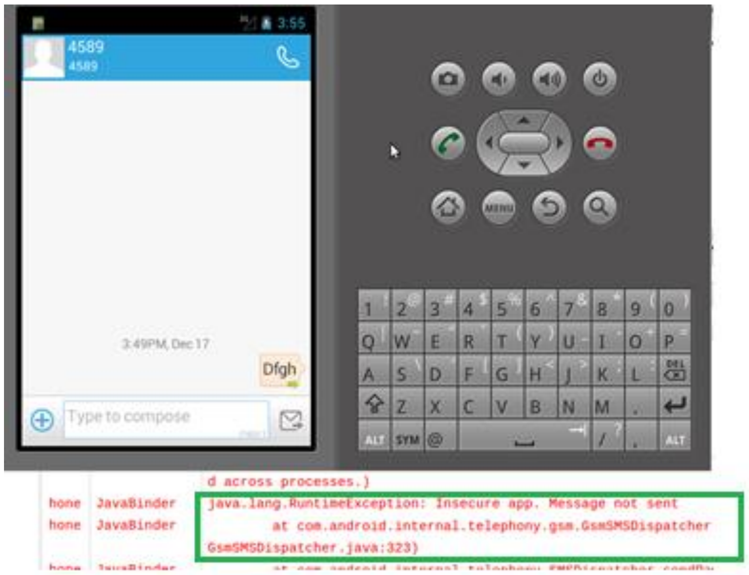
		runtime exception is thrown.
Frameworks/base/telephony/java (com.android.internal.telephony.cdma)	CdmaSMSDispatcher.java	Modified sendSMS method to throw runtime exception if data is Tainted.
Libcore/luni/src/main/java (org.apache.harmony.xnet.provider.jsse)	OpenSSLSocketImpl.java	Modified to throw runtime exception in method write if taint bit is not cleared.
Libcore/luni/src/main/java (libcore.io)	Posix.Java*	Identified sink points for reading/writing data to disk.

*We have not put Exceptions in Posix.java because it reads existing taints from filesystem and if we throw exception without handling them, system does not start.

Steps to test the code (SMS/MAPS):

- Go to Taint droid folder root and run the following commands in the mentioned order.
  build/envsetup.sh
  set_stuff_for_environment
  lunch 1
  make update-api
  make –j4
  emulator -kernel kernel-goldfish-xattr-2.6.29

- Once the emulator starts follow the below mentioned steps:
  - Install the GO SMS Pro application (apk is bundled with our code).
    - *adb install gosmspro.apk*
  - Go to Phone menu and start Taint Droid Notify Controller application and click on start.
  - Now start the GO SMS PRO application and do the following steps:
    Click on new message icon to start a new message
    Type destination number, something in the message body and click on send.
    Open the LogCat in eclipse IDE.
    Runtime exception is thrown with a message "Insecure App. Message not sent" (see the screen shot)
    Also in the emulator you can see that the message we tried to send is still in "sending" state.

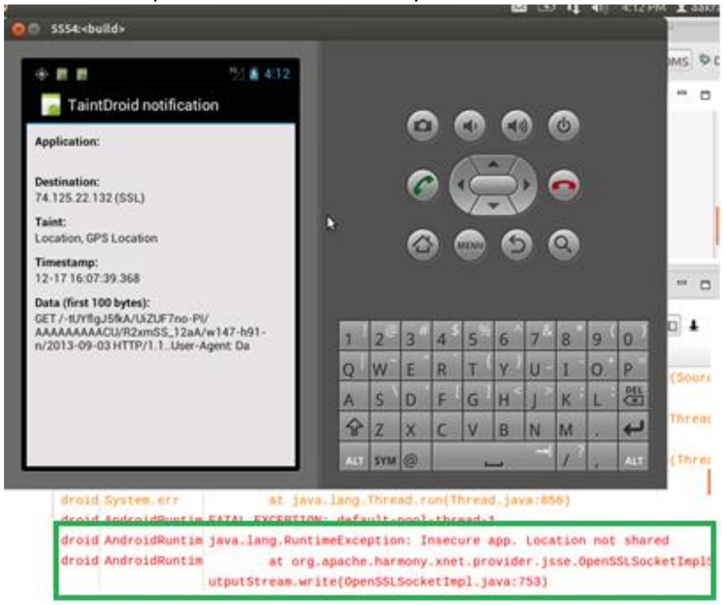


```
                    d across processes.)
hone  JavaBinder    java.lang.RuntimeException: Insecure app. Message not sent
hone  JavaBinder            at com.android.internal.telephony.gsm.GsmSMSDispatcher
                    GsmSMSDispatcher.java:323)
```

- Now start the Native messaging application (SMS Messaging) of the Android OS:
  Create a new message and click on send
  The message will be successfully sent and no notification will appear on screen.

- Test for Maps. Start the native MAPS application for android
  Before starting maps on emulator, set the GPS coordinates using the commands:
  telnet localhost 5554)
  geo fix <longitude> <latitude>

- Search for some location in Maps.
  - Taint droid notification will be seen for Location- GPS Location
  - LogCat will show a runtime exception with a message "Insecure App. Location not shared" (See the screen shot)



Application:

Destination:
74.125.22.132 (SSL)

Taint:
Location, GPS Location

Timestamp:
12-17 16:07:39.368

Data (first 100 bytes):
GET /-tUYflgJ5fkA/UiZUF7no-PI/AAAAAAAAACU/R2xmSS_12aA/w147-h91-n/2013-09-03 HTTP/1.1..User-Agent: Da

```
droid System.err                at java.lang.Thread.run(Thread.java:856)
droid AndroidRuntim java.lang.RuntimeException: Insecure app. Location not shared
droid AndroidRuntim         at org.apache.harmony.xnet.provider.jsse.OpenSSLSocketImpl$
                    utputStream.write(OpenSSLSocketImpl.java:753)
```

References

http://www.csc.kth.se/utbildning/kth/kurser/DD2460/sss12/assignments/workshop/students-slides/taintdroid.pdf
https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Enck.pdf