Discussion on "Systematic Detection of Capability Leaks in Stock Android Smartphones"

By William Ng

Android permission security system

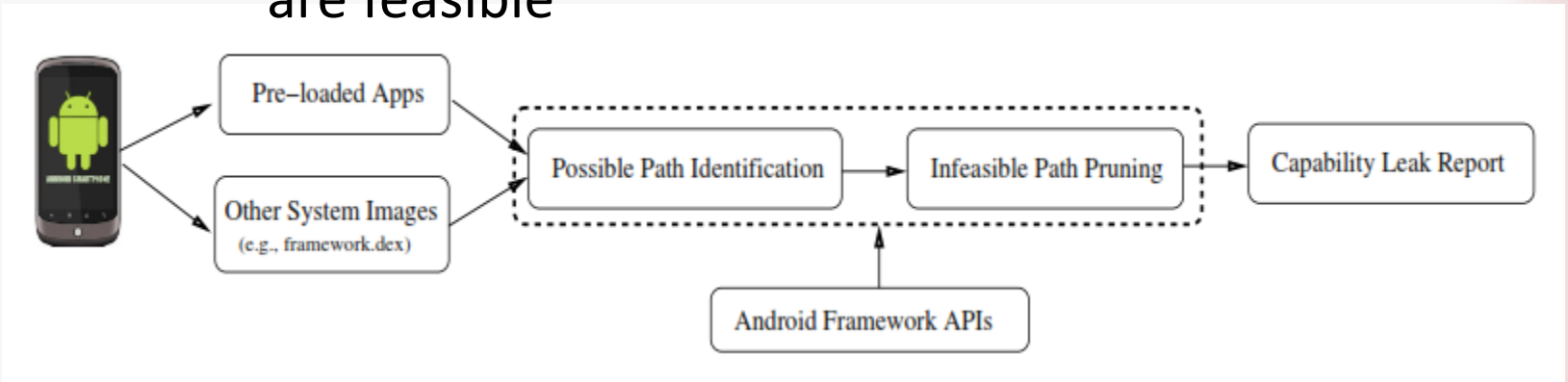
- Smartphone has been used to store and handle more and more personal data
- There is a need of a way to mediate the access to personal information or phone function.
  - Apple prompt users to approve the use of function at run-time, upon first access
  - Android requires application to request permission up-front

The breakdown of the permission system

- Capability leaks – App gain access to permission without actual requesting it
  - Explicit leaks: exploit publicly-accessbile interfaces or services
  - Implicit leaks: acquire or “inherit” permission from another application
- This paper analyzed the preloaded app to see if they caused any leaks

Explicit Capability Leak detection

- It involves 2 steps:
  1. Possible Path Identification
     - Identify possible paths from well-defined entry point (public interface) to some use of the capability
  2. Infeasible Path Pruning
     - Employs field- and path-sensitive inter-procedural data flow analysis to determine which of these paths are feasible



Possible Path identification – Control flow graph (wiki)

- A control flow graph (CFG) is a representation, using graph notation, of all paths that might be traversed through a program during its execution
  - Each node in the graph represents a basic block
  - Directed edges are used to represent jumps in the control flow

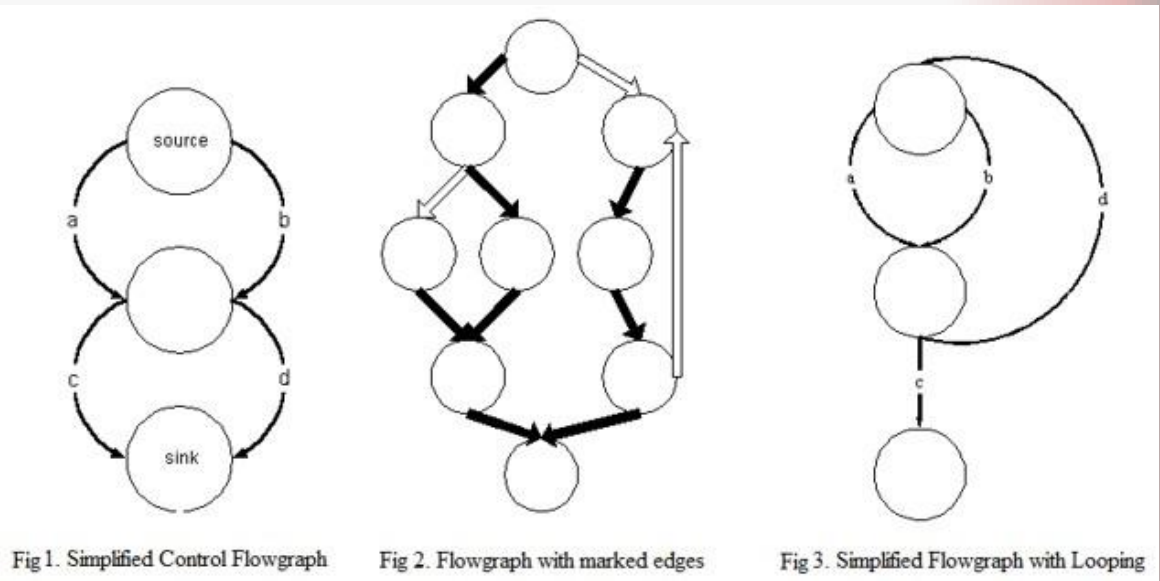


Fig 1. Simplified Control Flowgraph

Fig 2. Flowgraph with marked edges

Fig 3. Simplified Flowgraph with Looping

Possible Path identification – entry points

- There can be different type of components contained in the app, defined by the manifest. Each has a pre-defined interface to the rest of the system, which could be exploited.
- So they represents multiple entry points for CFG analysis.
- For example, onReceive would be an entry point for BroadcastReceiver class

Capability leak detection – dangerous API

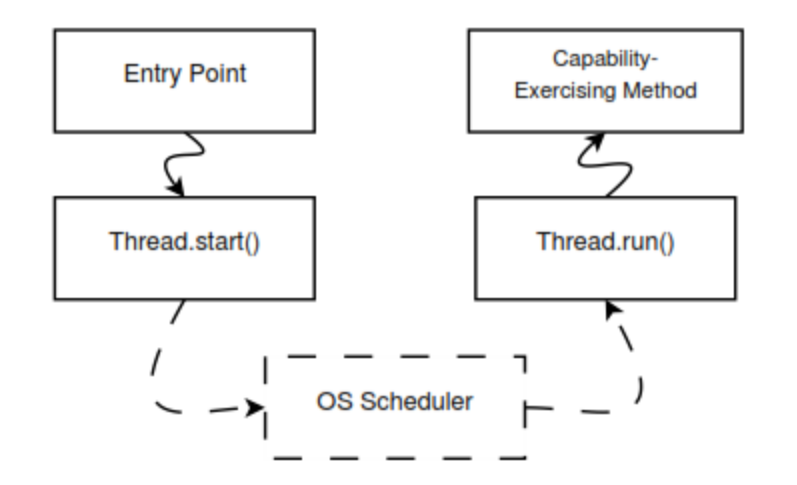
- Need to identify the list of Android APIs that might exercise the permission
- Each identified Android API call will be stored in the *dangerous* class.
- *Permission* class stored the information whether permission is checked.
- Capability leaks if it contains a dangerous call without checking the permission

Implementation issue with control flow graph – issue 1

- Issue 1: Indirect control-flow transfer instructions in Dalvik bytecode
  - Due to inheritance, often not possible to determine what concrete class a reference represents
  - Solution: when ambiguous reference encountered, consider all possible assignable class

Implementation issue with control flow graph – issue 2

- Issue 2: Android's event-driven nature
  - App execution passes through framework and emerge elsewhere



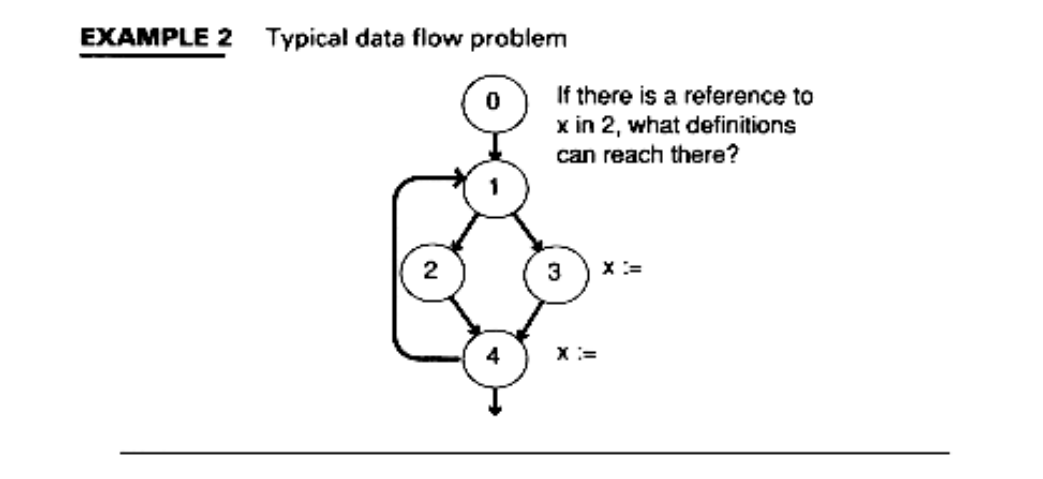
  - Solution: semantics are well-defined and understood, so they applied them to known callbacks and methods in the framework

Implementation issue with control flow graph – issue 3

- Issue 3: Applications may contain multiple entry points
  - Multiple components are defined in manifest file
  - Each component can potentially define multiple entry points accessible through the Binder IPC mechanism
  - Solution: run them all!

Feasible Path refinement – Inter-procedural data flow analysis

- Data flow analysis (wiki):
  - Gathering information about the possible set of values calculated at various points
  - It is trying to solve something like this:



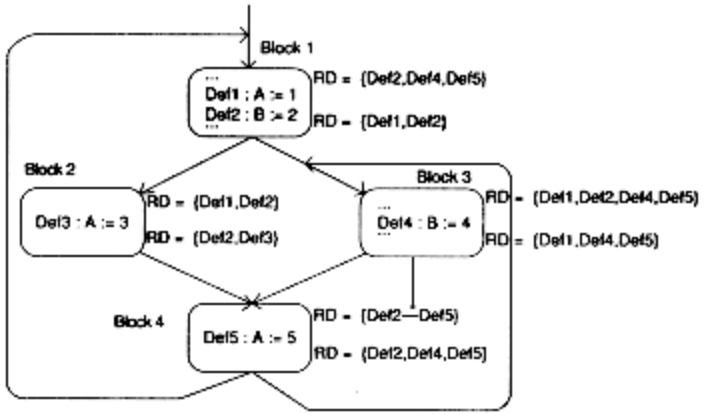
Feasible Path refinement – Inter-procedural data flow analysis

- Path of execution
  - modeled as set of program states, following one another
  - Transfer function that produce output states from input states
  - Algorithm converges on a solution

Feasible Path refinement – Inter-procedural data flow analysis

EXAMPLE 4 Reaching Definitions for Example 3

	Block 1	Block 2	Block 3	Block 4
FOR Loop	In = ϕ Out = {Def1,Def2}	In = ϕ Out = {Def3}	In = ϕ Out = {Def4}	In = ϕ Out = Def5
WHILE Loop Iteration 1	In = {Def5} Out = {Def1,Def2}	In = {Def1,Def2} Out = {Def2,Def3}	In = {Def1,Def2,Def5} Out = {Def1,Def4,Def5}	In = {Def1—Def5} Out = {Def2,Def4,Def5}
WHILE Loop Iteration 2	In = {Def2,Def4,Def5} Out = {Def1,Def2}	In = {Def1,Def2} Out = {Def2,Def3}	In = {Def1,Def2,Def4,Def5} Out = {Def1,Def4,Def5}	In = {Def1—Def5} Out = {Def2,Def4,Def5}
WHILE Loop Iteration 3	In = {Def2,Def4,Def5} Out = {Def1,Def2}	In = {Def1,Def2} Out = {Def2,Def3}	In = {Def1,Def2,Def4,Def5} Out = {Def1,Def4,Def5}	In = {Def1—Def5} Out = {Def2,Def4,Def5}



The values in the first set of Example 4 are the values computed using the FOR Loop and using the definitions of Gen above. In the third iteration of the WHILE Loop, the values are the same as for the second iteration; thus there are no changes and the final solution is found. The control flow graph is labeled with the final result.

What is implicit capability leak

- Permit an app to acquire or “inherit” permission from another app with the same signing key
- Misrepresent the capability available to an app
- Arise from an optional attribute in the manifest file “sharedUserId.” which allows multiple apps signed by the same developer certificate to share a user identifier.
- This system reports the exercise of an unrequested capability, which suspiciously has been requested by another app by the same author

Implicit capability leak – analysis

- Similar algorithm as explicit leaks detection but with a few changes
- The fundamental difference is that explicit capability leak detection assumes the caller of an app's exposed API is malicious, while implicit capability leak detection assumes the app itself might be malicious.
- Need to broaden the search to include the app's initialization:
  - Instance constructor: new-instance bytecode operation in dalvik
  - Class instructor or static initialization blocks
    - Could be invoked in variety of orders

Implicit capability leak – reporting

- Consider capability to be leaked if there is anyway to exercise the unrequested permission
- Union the permissions granted to each application with a given shared user identifier, which yields the set of permissions given to each of them.
- Report any implicitly leaked permissions contained within that set

Evaluation

- Studied eight smartphones from four vendors

Table 2. Eight studied Android smartphones

Vendor	Model	Android Version	# Apps
HTC	Legend	2.1-update1	125
	EVO 4G	2.2.2	160
	Wildfire S	2.3.2	144
Motorola	Droid	2.2.2	76
	Droid X	2.2.1	161
Samsung	Epic 4G	2.1-update1	138
Google	Nexus One	2.3.3	76
	Nexus S	2.3.3	72

Capability leaked

Permission	HTC						Motorola				Samsung		Google			
	Legend		EVO 4G		Wildfire S		Droid		Droid X		Epic 4G		Nexus One		Nexus S	
	E	I	E	I	E	I	E	I	E	I	E	I	E	I	E	I
ACCESS_COARSE_LOCATION	✓	✓	✓	✓	·	✓	·	·	✓	·	·	·	·	·	·	·
ACCESS_FINE_LOCATION	✓	·	✓	·	·	✓	·	·	✓	·	·	·	·	·	·	·
CALL_PHONE	·	·	·	·	·	·	·	·	·	·	✓	✓	·	·	·	·
CALL_PRIVILEGED	·	·	·	·	·	✓[1]	·	·	·	·	·	·	·	·	·	·
CAMERA	✓	·	✓	·	✓	·	·	·	·	·	·	·	·	·	·	·
DELETE_PACKAGES	✓[2]	·	✓[2]	·	✓[2]	·	✓[2]	·	✓[2]	·	✓[2]	·	✓[2]	·	✓[2]	·
INSTALL_PACKAGES	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
MASTER_CLEAR	·	·	·	·	·	·	·	·	·	·	✓	·	·	·	·	·
READ_PHONE_STATE	·	✓	·	✓	·	✓	·	·	✓	·	·	✓	·	·	·	·
REBOOT	·	·	✓	·	·	·	·	·	·	·	·	·	·	·	·	·
RECORD_AUDIO	✓	·	✓	·	✓	·	·	·	·	·	·	·	·	·	·	·
SEND_SMS	✓	·	✓	·	✓	·	·	·	·	·	·	·	·	·	·	·
SHUTDOWN	·	·	✓	·	·	·	·	·	·	·	·	·	·	·	·	·
Total	6	2	8	2	4	4	1	0	4	0	3	2	1	0	1	0

Stock android phone

- Phone image with close to stock android software has less capability leaks
- The stock android phone only have a single minor explicit leak that *com.svox.pico* could be tricked to remove another app *com.svox.langpack.installer*

Case studies – explicit capability leaks (without arguments)

- Samsung Epic 4G's has a preloaded app, com.sec.android.app.SelectiveReset, whose purpose is to display a confirmation screen that asks the user whether to reset the phone
- It has a class SelectiveResetReceiver to listen for SELECTIVE_RESET Intent.
- After it receives the intent, it open GUI and wait for user to confirm

Case studies – explicit capability leaks (without arguments)

- Once that is done, it starts a service SelectiveResetService which broadcasts an intent SELECTIVE_RESET_DONE
- SelectiveResetReceiver class listens for this Intent and then calls CheckinService.masterClear()
- But this intent can be initiated by malicious app because

Case studies – explicit capability leaks (without arguments)

- But this intent can be initiated by malicious app because SelectiveResetReceiver is defined in the manifest file
- Similar leaks for REBOOT and SHUTDOWN is observed on HTC EVO 4G; or capability FREEZE is exposed by a system app

Performance management - speed

- Computer: AMD Athlon 64 X2 5200+ machine with 2GB of memory and a Hitachi HDP72502 7200 rpm hard drive
- Data is averaged over 10 runs

Vendor	Model	Processing Time	# Apps
HTC	Legend	3366.63s	125
	EVO 4G	4175.03s	160
	Wildfire S	3894.37s	144
Motorola	Droid	2138.38s	76
	Droid X	3311.94s	161
Samsung	Epic 4G	3732.56s	138
Google	Nexus One	2059.47s	76
	Nexus S	1815.71s	72

Performance Measurement – false positive and negative

- All the leaks reported by the system is verifed manually, so there is no false positive.
- There is lack of ground truth and the approach is conservative, so the author is confident that the false negatives number is low.

Last words from authors

- Rather than relying on app creator to diligently check for capability leaks, Android framework could also be improved to mediate app interaction and defend the integrity of capability.
- Possibly to make this into a validator tool to help app developer.

Future works from authors

- Expand to handle native code, but not just Dalvik code
- Expand to handle app-defined permission
- Expand to analyze third party app developers but not just preloaded app