# Prioritized Analysis of Inter-App Communication Risks[*]

Fang Liu[1], Haipeng Cai[2], Gang Wang[1], Danfeng (Daphne) Yao[1], Karim O. Elish[3], and Barbara G. Ryder[1]

[1]Department of Computer Science, Virginia Tech

[2]School of Electrical Engineering and Computer Science, Washington State University

[3] Department of Computer Science, Florida Polytechnic University

fbeyond@cs.vt.edu, hcai@eecs.wsu.edu, {gangwang,danfeng}@cs.vt.edu, kelish@flpoly.org, ryder@cs.vt.edu

## ABSTRACT

Inter-Component Communication (ICC) enables useful interactions between mobile apps. However, misuse of ICC exposes users to serious threats such as intent hijacking/spoofing and app collusions, allowing malicious apps to access privileged user data via another app. Unfortunately, existing ICC analyses are largely incompetent in both accuracy and scale. This poster points out the need and technical challenges of prioritized analysis of inter-app ICC risks. We propose MR-Droid, a MapReduce-based computing framework for *accurate* and *scalable* inter-app ICC analysis in Android. MR-Droid extracts data-flow features between multiple communicating apps and the target apps to build a large-scale ICC graph. Our approach is to leverage the ICC graph to provide contexts for inter-app communications to produce precise alerts and prioritize risk assessments. This process requires large app-pair data, which is enabled by our MapReduce-based program analysis. Our initial extensive experiments on 11,996 apps from 24 app categories (13 million pairs) demonstrate the scalability of our approach.

## 1. Introduction

Inter-Component Communication (ICC) is an important mechanism for app-to-app communications on Android. It links the components of different apps via messaging objects (or *Intents*). While ICC contributes greatly to the development of rich third-party applications, this communication model has become a predominant security attack surface for Intent hijacking, Intent spoofing and app colusions. According to a recent report from McAfee Labs [1], app collusions are increasingly prevalent on mobile platforms.

To assess ICC vulnerabilities, various analytics methods have been proposed. However, most of them perform analysis for one *individual* app at a time, ignoring its feasible communication context with other apps. As the consequence, they provide conservative risk estimations, producing a high number of (false) alarms [2, 3]. A more effective approach is inter-app analysis that consider ICCs across two or more apps. This allows researchers to gain empirical contexts on the actual communications between apps and produce more relevant alerts. However, existing solutions are largely limited in scale due to the high complexity of pair-wise components analyses. They were either applied to a much smaller set of apps (a few hundred only, versus a few thousand in single-app analyses), or a small set of inter-app links.

A recent study PRIMO reported ICC analyses of a large pool of apps [6]. Its goal is to approximate the likelihoods of ICC communications between app pairs through learning from training data. However, PRIMO is not designed for ICC risk analysis, thus it does not provide classification mechanisms for security. Moreover, PRIMO runs on a single workstation. Our evaluation shows that running PRIMO with 10K apps requires over 40GB memory even with highly compressed metadata. Using a single machine is not a practical solution to analyze market-wide apps with the space complexity being $O(N^2)$.

In this paper, we point out the need and technical challenges of prioritized analysis of inter-app ICC risks. We propose MR-Droid, a MapReduce-based parallel analytics system for *accurate* and *scalable* ICC risk detection. Our goal is to evaluate ICC risks based on an app's inter-connections with other real-world apps and efficiently identify high-risk pairs. Our intuition is that an ICC pair is of high-risk, not only because one of the apps is vulnerable, but more importantly it potentially communicates with other apps. To achieve this goal, we construct a large-scale *ICC graph*, where each node is an app component and the edge represents the corresponding inter-app ICCs[1]. To gauge the risk level of the ICC pairs (edge weight), our approach is to extract various features based on app flow analyses that indicate vulnerabilities. For instance, we can examine whether the ICC pair is used to pass sensitive data, or escalate permissions for another app. With the ICC graph, we can further *rank* the risk level of a given app by aggregating all its ICC edges with other apps.

To scale up the system, we implement MR-Droid with a set of new MapReduce algorithms atop the Hadoop framework for constructing the ICC graph. MapReduce has been used in various areas (e.g., data leak detection [5]). We independently exact ICC sources and sinks from each app, and

[*]This work has been supported in part by DARPA APAC FA8750-15-2-0076 and ARO YIP W911NF-14-1-0535.

*CODASPY'17 March 22-24, 2017, Scottsdale, AZ, USA*

© 2017 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4523-1/17/03.

DOI: http://dx.doi.org/10.1145/3029806.3029843

[1]We use *ICC graph* to represent inter-app communications throughout the paper.

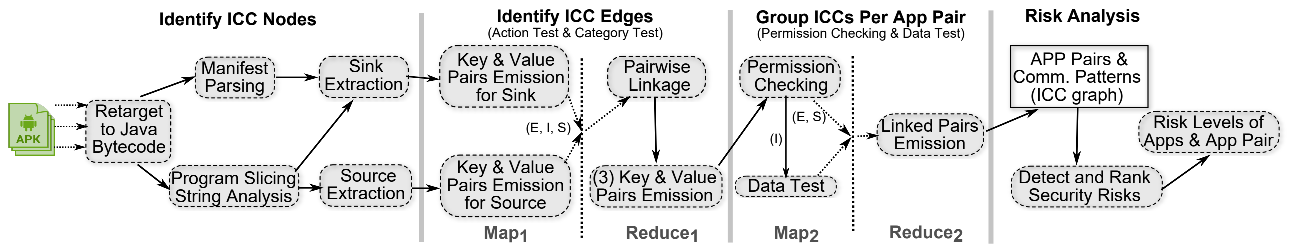
Identify ICC Nodes
Identify ICC Edges
(Action Test & Category Test)
Group ICCs Per App Pair
(Permission Checking & Data Test)
Risk Analysis
APK
Retarget to Java Bytecode
Manifest Parsing
Program Slicing String Analysis
Sink Extraction
Source Extraction
Key & Value Pairs Emission for Sink
Key & Value Pairs Emission for Source
(E, I, S)
Pairwise Linkage
(3) Key & Value Pairs Emission
Permission Checking
(I)
(E, S)
Data Test
Linked Pairs Emission
APP Pairs & Comm. Patterns (ICC graph)
Detect and Rank Security Risks
Risk Levels of Apps & App Pair
Map1
Reduce1
Map2
Reduce2

Figure 1: Our workflow for analyzing Android app pairs with MapReduce. The dashed vertical lines indicate redistribution processes in MapReduce. E, I, S represent explicit edge, implicit edge, and sharedUserId edge respectively.

jointly analyze those with matched source-sink pairs with MapReduce. The high-level parallelization from MapReduce allows us to analyze millions of app pairs within hours using commodity servers.

We evaluate ICC graph construction on a large set of 11,996 Android apps collected from 24 major Google Play categories (13 million ICC pairs). Our system is highly scalable. With 15 commodity servers, the entire process took less than 25 hours for an average of only 0.0012 seconds per app pair. More importantly, our runtime experiment shows the computation time grows near-linearly with respect to the number of apps.

# 2. Models and Methodology

In this section, we describe the threat model and computational goal of our work. Then we give an overview of our approach.

## 2.1 Threat Model

Our work focuses on security risks caused by inter-app communications realized through ICCs, covering three most important classes of inter-app ICC security risks.

- **Intent hijacking.** An Intent sent by an app via an implicit ICC may be intercepted (hijacked) by an unauthorized app. This threat scenario, referred to as Intent hijacking, can be categorized into three subclasses according to the type of the sending component: broadcast theft, activity hijacking, and service hijacking, as introduced in [2].
- **Intent spoofing.** By sending Intents to exported components of a vulnerable app, an attacker can spoof the vulnerable (receiving) app to perform malicious actions. Intent spoofing can be classified into three subclasses by the type of the receiving component: malicious broadcast injection, malicious activity launch, and malicious service launch [2].
- **Malware collusion.** Through inter-app ICCs, two or more apps may collude to perform malicious actions that none of the participating apps alone would be able to. Malware collusion can result in disguised data leak and system abuse.

## 2.2 Computational Goal

Our computational goal is two-fold.

1. Build a complete inter-app ICC graph and identify all communication app pairs for a set of apps to provide the communication context (i.e., the neighbor set) for each one.
2. Further perform neighbor-aware inter-app security analysis on top of the ICC graph and rank the apps and app pairs with respect to their risk levels.

The framework for building the ICC graph has to be able to process very large-scale real world apps efficiently. There is a potentially higher risk of missing true risk warnings if limited size of communication context are considered (e.g., a highly vulnerable app may be declared safe when only a few external apps are analyzed).

To rank the apps and app pairs with respect to their risk levels, one has to carefully represent the apps' communication context given the ICC graph. Different threat models may require different communication context for ranking.

## 2.3 Overview

As shown in Figure 1, our workflow involves two major phases: distributed ICC mapping for building ICC graph in large-scale and neighbor-based risk analysis to rank the apps according to their risk levels.

Distributed ICC mapping is implemented with MapReduce on top of Hadoop for scalability. It has three steps: step 1. IDENTIFY ICC NODES, 2. IDENTIFY ICC EDGES, and 3. GROUP ICCS PER APP PAIR.

1. In IDENTIFY ICC NODES, we extract the attributes of sources and sinks from apps. Sources are extracted from the attributes in outbound Intents. Sinks are extracted from the exported components in the manifest or from dynamic receivers created in the code.
2. IDENTIFY ICC EDGES is the first MapReduce job which identifies edges between communicating sources and sinks. The MapReduce job transforms the source and sinks into ⟨*key, value*⟩ pairs, which enable parallel edge finding.
3. GROUP ICCS PER APP PAIR is the second MapReduce job which performs the data test and permission checking, and identifies and groups edges belonging to the same pair.

In addition, we balance the workload among all the nodes in the MapReduce cluster for the best performance. Unbalanced workload highly impacts the performances because most of the nodes are idle while waiting for the nodes with the heavy workload. We address this problem by adding a tag before each key emitted by the Map function. The tag helps to divide the large amount of key-value pairs, which should be sent to one reducer, into $m$ parts feeding $m$ reducers.

Neighbor-based risk analysis is to utilize the ICC mapping results (ICC graph) from the previous phase to compute key ICC link features, and then uses the features to rank security risks for each app (for hijacking and spoofing attacks) and app pair (for collusion attacks). Different features may be used for different attack models.

- HIJACKING We assess the possibility of an intent to be hijacked be using the features with outbound links (e.g., Number of outbound edges with data).

- SPOOFING We assess the possibility of component to be spoofed using the features with inbound links (e.g., Number of outbound edges with data).
- COLLUSION All communicating pairs are analyzed together to assess their collusion possibility. Both inbound and outbound features are involved for the assessment.

Given an app or app pair, our analysis first computes its risk with respect to individual features, and then aggregates them to obtain an overall risk value. The risk for an individual feature is ranked based on the feature value distribution of all apps/app pairs. An app's individual feature risk is higher, if the corresponding value is larger than other apps.

## 3. Evaluation

We implemented our system with native Hadoop MapReduce framework. The input is the ICC sources and sinks extracted from individual apps using IC3 [7]. We modified IC3 to accommodate the MapReduce paradigm. The Hadoop system is deployed on a 15-node cluster. Each node has two quad-core 2.8GHz Xeon processors and 8GB RAM.[2]

**Datasets**

For our evaluation, we apply our system to 11,996 most popular free apps from Google Play. We select the top 500 apps from each of the 24 major app categories (4 apps were unavailable due to bugs in program analysis). We downloaded the apps in December 2014 with an Android 4.2 client.

**Risk Assessment**

We apply our system to the collected app dataset. The resulting ICC graph contains 38,134,207 source nodes, 26,227,430 sink nodes and 75,123,502 edges. On the per-app level, there are in total 12,986,254 app pairs that have at least one ICC link. Each app averagely connects with 1185 external apps (9.9% of all apps). For non-connected app pairs, we can safely exclude them during the security analysis.

Our security analysis is focused on all potential security risks related to Intent hijacking, Intent spoofing and app collusion. We quantify and rank security risks into as categorical risk levels. In total, our system identified 150 high-risk apps, 1,021 medium and 10,825 low risk apps. We report the runtime performance of MR-Droid next. Our complete evaluation result will be reported in a full version soon.

**Runtime of MR-Droid**

We analyze the runtime performance of MR-Droid. Figure 2 depicts the time cost of our MapReduce pipeline ($y$ axis) as the number of apps increases ($x$ axis). Overall, the result shows that our approach is readily scalable for large-scale inter-app analysis. The running time of ICC node identification appears to dominate the total analysis cost, yet its growth is linear with the number of apps. In addition, given the sparse nature of the ICC graph (rarely does an app communicate to all apps), we manage to achieve near-linear complexity for edge identification and grouping ICCs. In total, it takes 25 hours to perform the complete analysis on 13 million ICC pairs for 12K apps. Currently, our cluster has 15 nodes. We anticipate that increasing the cluster size would further speed up the inter-app ICC analysis.

[2]The algorithms can also be implemented with Spark with faster in-memory processing. It will require much larger RAMs.

As a baseline comparison, we evaluated the performance of IccTA [4], a non-distributed inter-app ICC analysis system. IccTA needs to first combine two or more apps into one app and then perform ICC analytics. We evaluate IccTA with 57 randomly selected real world apps on a workstation (80GB RAM). It took IccTA over 200 hours to analyze all the apps. We estimate that processing 200 apps with IccTA would take about 18,000 hours, making it impractical for analyzing market-scale apps.

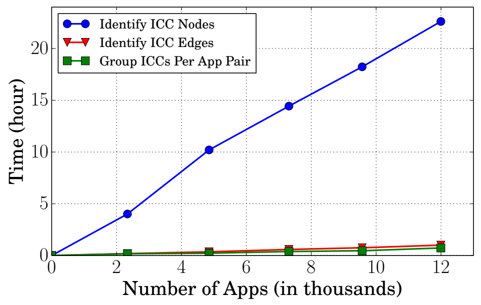

Figure 2: Analysis time of the three phases in our approach.

## 4. Conclusion and Future Work

In this paper, we pointed out the need and technical challeges of prioritized analysis of inter-app ICC risks. We presented the design and implementation of MR-Droid, a MapReduce pipeline for large-scale inter-app ICC risk analyses. By constructing ICC graphs with efficient parallelization, our system enables highly scalable inter-app security analysis for accurate risk prioritization.

Because of the lack of ground truth on the empirical data, we will further devote substantial future efforts to manually inspecting the apps for validation.

## 5. References

[1] Mcafee labs threats report. http://www.mcafee.com/us/resources/reports/rp-quarterly-threats-may-2016.pdf, 2016.

[2] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner. Analyzing inter-application communication in Android. In *MobiSys*, 2011.

[3] K. O. Elish, D. D. Yao, and B. G. Ryder. On the need of precise Inter-App ICC classification for detecting Android malware collusions. In *MoST, IEEE S&P*, 2015.

[4] L. Li, A. Bartel, T. F. Bissyandé, J. Klein, Y. Le Traon, S. Arzt, S. Rasthofer, E. Bodden, D. Octeau, and P. Mcdaniel. IccTA: Detecting inter-component privacy leaks in Android apps. In *ICSE*, 2015.

[5] F. Liu, X. Shu, D. Yao, and A. R. Butt. Privacy-preserving scanning of big content for sensitive data exposure with mapreduce. In *CODASPY*, 2015.

[6] D. Octeau, S. Jha, M. Dering, P. McDaniel, A. Bartel, L. Li, J. Klein, and Y. Le Traon. Combining static analysis with probabilistic models to enable market-scale Android inter-component analysis. In *POPL*, 2016.

[7] D. Octeau, D. Luchaup, M. Dering, S. Jha, and P. McDaniel. Composite constant propagation: Application to Android inter-component communication analysis. In *ICSE*, 2015.