网络出版时间：2017-03-04 17:40:08
网络出版地址：http://kns.cnki.net/kcms/detail/11.2127.TP.20170304.1740.016.html

paraIntentFuzz:安卓应用漏洞的并行化模糊测试方法

李川[1]，刘宝旭[2]
LI Chuan[1], LIU Baoxu[2]
1.福州大学 数学与计算机科学学院，福州 350000
2.中国科学院 信息工程研究所，北京 100093
1.School of Mathematics and Computer Science, Fuzhou University, Fuzhou 350000, China
2.Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

LI Chuan, LIU Baoxu. ParaIntentFuzz: Android applications parallel fuzzing system. Computer Engineering and Applications

Abstract: Permission leakage is a common kind of vulnerability among Android applications, this kind of vulnerability can lead to serious security problem. By fuzzing the Intent to discover the expose of components and find the permission leakage from the exposed components is an efficient method to mine permission leakage. However, existing works based on Intent Fuzz to test this kind of vulnerability is only running on single machine, which leads to low availability. In this paper, a parallel fuzzing system based on dynamic task distribution, named paraIntentFuzz, is implemented. It first extract extra information from application by static analysis and then construct Intent commands. After sending commands to target application via Drozer, paraIntentFuzz can effectively fuzz the target application, the system is deployed on four computers. With paraIntentFuzz, we analyzed 10064 Android applications and find 7367 of them have permission leakage problem.

Key words: Permission leakage; vulnerability; parallel ; Android

摘要：权限泄露是安卓应用中较为普遍存在的一类漏洞，可导致较为严重的安全问题，例如“串谋提权”等。通过 Intent 模糊测试技术发现暴露的组件，是挖掘权限泄露漏洞的有效方法。但是现有 Intent 模糊测试技术仅限于单机运行，效率低下。本文提出一种基于动态任务分配的并行模糊测试方法 paraIntentFuzz。该方法通过静态分析提取出安卓应用的 extra 信息并构造 Intent 命令，通过 Drozer 工具给目标应用发送命令，实现了独立的模糊测试，并部署到 4 台机器上。使用它分析了 10064 个 Android 个应用，最后发现有 7367 个应用存在权限泄露的问题。

关键词：权限泄露；漏洞挖掘；并行；Android
doi:10.3778/j.issn.1002-8331.1609-0195 **中图分类号**：TP393

1 引言

Android 操作系统是近几年市场占有率最高的智能手机操作系统，据腾讯的 Bugly 2016 年 4 月 8 日的报道[1]，目前仅腾讯应用宝上面国内的应用数量就达到了 300 万， 在此背景下，Android 应用的安全性显得十分重要。

Android 应用的很多重要的功能都是基于组件实现的，开发人员基于它们实现应用的各种功能，这些组件可能会包含高权限的敏感操作[2]，对外暴露可能会造成权限泄露等危害。

模糊测试（Fuzz）[3]是一种对权限泄露进行挖掘的一种有效的方法，但是目前基于 Intent 模糊测试的 Android 应用权限泄露测试系统都是单机运行，而且有的在发送的 Intent 命令中没有加入 extra 信息、有的没有加入很好的日志处理[4]。这些系统效率比较低，可用性有待改善。如何高效的获取目标应用组件间通信的 extra 信息、如何搭建高性能以及可扩展的并行大规模 Android 应用模糊测试系统、如何对实验结果进行分析等问题都是有挑战性也是很值得研究的问题。

基金项目：国家高技术研究发展计划(863 计划) 课题（编号：2015AA017202）
作者简介：李川(1991－),男,硕士研究生，研究领域为 Android 漏洞挖掘;刘宝旭(1972－),男，博士，研究员，研究领域为网络与信息安全、攻防对抗、网络安全评测技术等。E-mail: liubaoxu@iie.ac.cn

本文实现了一个权限泄露漏洞挖掘系统paraIntenFuzz。paraIntenFuzz具有如下3个创新点：1、基于云计算平台实现并行的分析架构；2、提出了一种高效的extra信息静态获取方法，提高分析精确性；3、集成了动态的分析任务分配技术，提高分析效率。

我们用paraIntentFuzz完成了对10064个应用模糊测试分析，耗时216小时10分，发现7367个应用存在不同程度的权限泄露问题，泄露权限48种。

本文第1节引言介绍基于Intent Fuzz的Android权限泄露漏洞挖掘的研究现状，并提出本文的总体思路；第2节介绍了相关的研究工作；第3节介绍了相关的Android背景知识；第4节介绍了paraIntentFuzz设计的关键技术；第5节是实验测试及对实验数据的分析；第6节对工作进行总结和展望。

2 相关研究

2012年nccgroup公司推出的Intent Fuzzer[5]将测试手机所有应用的Activity、Broadcast、Provider、Service分类列出，然后对他们发Intent命令。但不同应用的同一类型的组件都混在了一起，发送的Intent中没有加入extra信息，也没有对日志信息进行很好的处理。

MindMac通过改进nccgroup的Intent Fuzzer开发出IntentFuzzer[6]，将手机中的所有应用以列表的形式呈现，选中某一个应用之后可以选择该应用的Activity、Broadcast Receiver、Service，然后对其进行模糊测试。改进之处在于可以知道当前测试的是那个应用的组件，但是发送的Intent中依然没有加入extra信息，也没有很好的日志处理。

诸葛建伟等人的IntentFuzzer[7]，采用Drozer来做作为命令的发送端，并在extra获取函数处插桩获取组件间通信[8]的extra信息。由于加入了extra信息，使程序能够执行更多的路径[9]，能发现更多的权限泄露。但是通过插桩来动态地获知应用的extra信息会比较耗时，而且他们的IntentFuzzer也是基于单机并没有扩展到多机并行。

3 背景知识

3.1 Android中的权限泄露漏洞

Android系统基于安全性的考虑而引入了权限机制[10]。通过Android的网站[11]可知Android系统中一共有100多种权限，并且不同的版本的Android系统可以使用的权限数目可能不一样。Android的权限一共分为4种级别[12]：normal、dangerous、signature、signatureOrSystem，Android应用在执行某些特权操作时需要申请相应的权限，这些操作通常实现在Android组件之上。Android四大组件包括Activity（活动）、Service（服务）、Content Provider（内容提供者）和Broadcast Receiver（广播接收器）[13]，四大组件的关系如下图1所示。从用户的角度，与之直接产生交互的是Activity，四大组件中的Activity、Service、Broadcast Receiver通过在AndroidManifest.xml[14]中注册IntentFilter，接收Intent，完成组件间通讯。随着Android市场的繁荣，越来越多的Android应用被开发者开发出来，但是由于开发者水平参差不齐，开发者的软件可能会存在各种问题，例如包含敏感操作的组件对外暴露，进而导致权限的泄露，因此从暴露的组件中挖掘Android应用中的权限泄露漏洞是一个有效的方法。

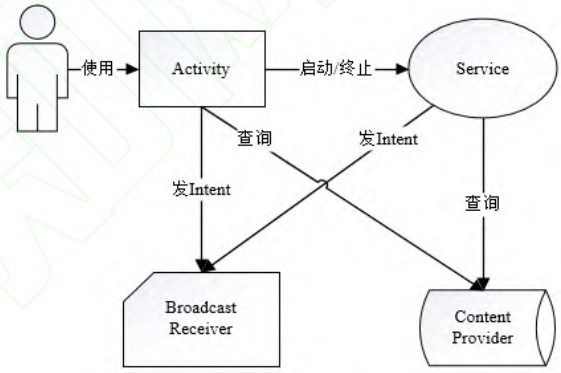

图1 Android各组件关系图

3.2 Intent机制和extra信息

Intent是Android中的一个抽象概念[15]，它用来描述将要执行的一些操作，Activity、Service、Broadcast都可以接收到Intent后启动而执行相应的操作。Intent中有一个属性extra，它可以看做是一个附加信息的集合，它里面可以存放int、String等8种基本Java数据类型以及可序化对象[16]。在Activity、Service、Broadcast这3个组件接收Intent的时候可以从Intent中获取extra信息，而extra信息对程序的控制流程是至关重要的，比如在程序中判断从extra中获取出的某一个变量的值不为空才接着执行某一些操作。所以为了让程序触发更多的操作，需要尽可能为Intent提供目标组件所需要的extra。

4 paraIntentFuzz的设计

本文使用实验室VARAS系统中基于OpenStack云平台的虚拟机集群，设计和实现了对Android应用

并行分析的框架——paraIntentFuzz。

4.1 基于云计算平台的并行分析架构

paraIntentFuzz 结构设计如下图 2 所示，具有两类结点：服务器结点和运行结点。服务器负责动态地向运行结点分配分析任务（服务器结点不仅负责调度、任务分配等工作，本身也充当一个运行结点运行模糊测试任务）。运行结点用来对目标应用进行模糊测试任务。具体包括：目标应用的预处理、命令的生成、命令的执行、执行结果的处理等步骤。

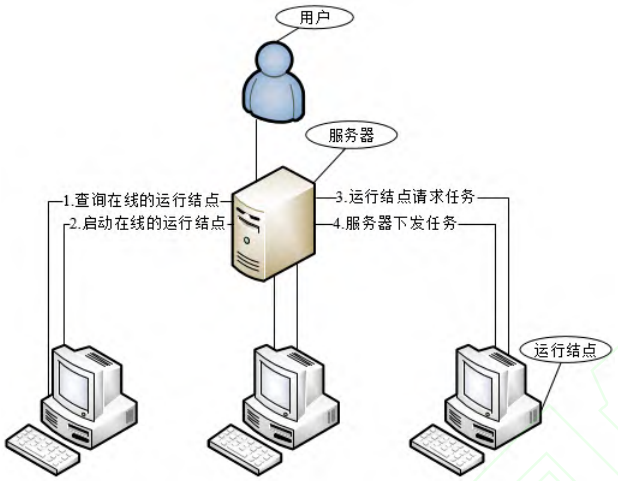

图 2 paraIntentFuzz 结构设计

为了系统能够在更短的时间内完成大批量的任务，在 paraIntentFuzz 中设计和实现了高效的并行模糊测试架构。由一个服务器结点和多个运行结点所组成，执行流程分为以下几个步骤：

1.服务器结点作为控制端查询任务队列中是否有任务，若有任务则查询在线的运行结点。

2.服务器结点下发任务给每一个在线的运行结点，然后启动在线的结点。

3.运行结点启动后会开始执行模糊测试，运行结点任务完成后会向服务结点请求新的任务进行分析。

4.服务器下发任务。

当客户端请求任务后一定时间后仍然没有任务分配时，客户端回传执行的结果给服务端。

4.2 轻量级的 extra 信息静态获取方法

获取 extra 的信息的方法分动态和静态两类方法，动态的方法有前文所述的诸葛建伟等人的工作，静态方法有 IC3[17]。但是 IC3 比较耗时且获得的 extra 值不包含变量的类型，只有变量名，无法知道它的类型。为了以更快、更准确地地获取应用中的 extra 信息，本文提出了一种高效的 extra 信息静态分析方法，不仅能获得 extra 信息的变量名，而且还知道变量的类型。获取 extra 信息的 Python 伪代码如下：

算法 1：获取应用的 extra 信息

输入：目标应用的绝对路径

输出：当前应用的所有 extra 信息

```
1:  getSmaliFiles(path)
2:  f = ReadEachSmaliFile()
3:      f.readline() #读取 smali 文件的每一行
4:          while line:
5:              if 方法的结尾:
6:                  初始化中间变量
7:              else:
8:                  if 变量定义或变量操作：
9:                      保存或更新变量的值
10:                 elif 新建 Intent 对象:
11:                     将 Intnet 名字放入结果中
12:                 elif put 或 get 函数操作 extra 信息
13:                     根据 Intent 名字将 extra 放入
                        结果中，表示该 Intent 的 extra
14:             line = f.readline() #读下一行
15: return result
```

算法先使用 Apktool 反汇编目标应用得到 smali[18] 文件，第 2 行函数会递归的遍历给定路径及其子目录下面的所有文件夹中的 smali 文件，第 3 行为读取 smali 文件的内容，对 smali 文件的每一行处理过程为：第 5 行判断如果是方法的结尾，则将中间变量初始化，第 8 行判断如果是变量的定义或操作，则将该变量的值保存下来。第 10 行判断如果是在新建一个 Intent，则将该 Intent 名字保存到结果中。第 12 行判断如果是往 extra 中放入数据或者从 extra 中取数据，则根据当前的 Intent 名字将该 extra 值放入结果中。分析完成后将结果返回。

4.3 动态的任务分配技术

为了目标任务在最短的时间内完成分析，我们提出了动态任务分配技术，其核心思想是预估每个应用的分析时间，然后做到有效的调度，提高每个运行结点的利用率。通过对单个应用分析时间的实验统计，我们发现每个应用的分析时间主要受应用的大小、暴露的组件的数目的影响即应用越大、组件越多通常会更耗时。如果采用静态地为每一个运行结点分配个数一样多的任务，可能造成每一个结点负载不均衡，进而导致完成分析需要更多的时间。基于此我们提出了如下动态任务调度算法的 Python 伪代码：

算法 2：任务调度算法

```
1:  sock = socket.socket()
2:  sock.bind()
3:  while True:
4:      taskList = getApkList()#获取任务列表并预测分析时间
5:      initialTaskDevide()
6:      while True:
7:          if 任务队列中还有任务：
8:              监听来自运行结点的任务请求并响应
9:          else:
10:             time.sleep(30)
11:             break
```

算法中第 1-2 行是初始化操作，通过调用 socket() 函数获得 sock 变量，由它来完成 socket 连接的任务。第 3-11 行是一个 while 循环。第 4 行的 getApkList() 获得任务队列 taskList，它会将所有的应用按大小降序排序，若大小取整后相同（如 2 个应用分别为 3.1MB、3.2MB，则取整后都是 3MB），则按暴露的组件数目降序排序，得到所有应用的名字的列表 taskList 并将任务编号 0~队列长度-1，分配任务的时候就从编号小的开始分配。第 5 行根据任务列表中的任务数据量来给每台机器分配 2 个任务（若不够 8 个则只为每台分配 1 个，若连小于 4 个为 k 个，则只为 k 个子结点每台分配 1 个任务）。第 8 行监听运行结点的连接，如果有新连接到就开启一个线程用来传输任务给运行结点，这样就可以传输的同时也监听来自其他运行节点的任务请求。

4.4 系统实现

4.4.1 paraIntentFuzz 整体实现

paraIntentFuzz 整体结构如上面图 2 所示，当需要执行分析任务时用户会首先开启服务器结点，然后开启在线的运行结点，然后就向运行结点分发任务。本节主要介绍每一个运行结点上 paraIntentFuzz 执行流程，paraIntentFuzz 的执行流程如下图 3 所示：

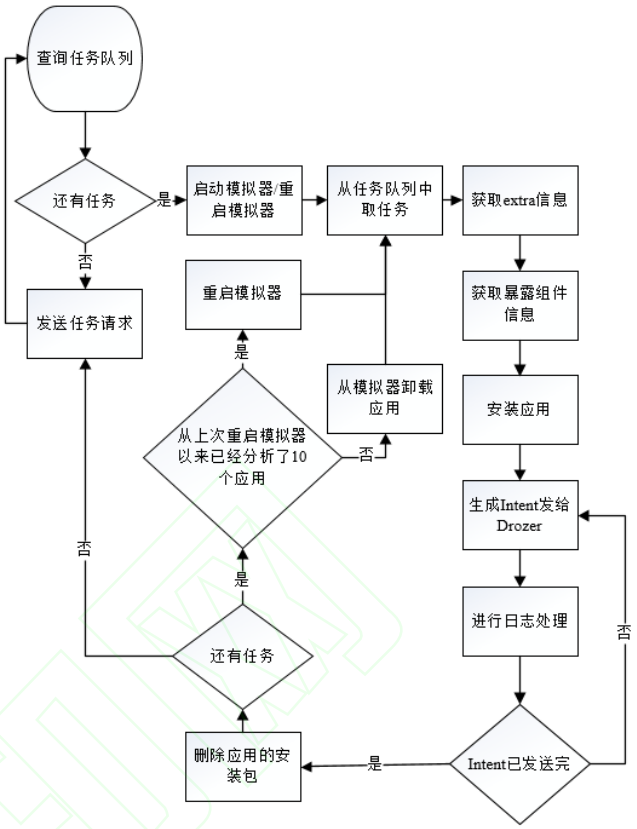

图 3 paraIntentFuzz 执行流程图

运行结点上安装了 JDK、Android SDK、Drozer[19]、Python 等分析所用的软件。通过对目标应用进行分析生成命令，然后发给 Drozer，Drozer 解析后发给代理应用，代理应用再发给目标应用，目标应用解析并执行命令，然后待命令执行完后获得模拟器中应用执行的日志信息并对日志信息进行分析以获得暴露的权限信息。

运行结点运行之后会查询任务队列，若队列中没有任务则向服务器端发送任务请求，若有则启动模拟器然后从任务队列中获取任务并开始执行模糊测试任务，执行如下过程：

1.使用 Apktool 反汇编目标应用得到 smali 代码然后从 smali 中获取 extra 信息。

2.从目标应用的 AndroidManifest.xml 文件中获取暴露的组件信息。

3.安装目标应用到模拟器中。

4.生成 Intent 命令并发给 Drozer，Drozer 会通过代理应用发给目标应用，目标应用解析并执行命令，然后目标应用解析完 Intent 命令后进行日志处理，日志处理完后会发送下一条 Intent 命令，一直这样重复直到所有命令都发送完毕。

5.删除目标应用的安装包。

6.判断是否还有任务，若没有则给服务器发送任

务请求，若有则判断从模拟器启动以来是否已经分析了 10 个应用，若从模拟器启动以来是否已经分析了 10 个应用则重启模拟器，否则卸载刚测试过的应用，进行下一个应用的测试。

7.接着执行 1~6 的任务执行流程。

整个框架的运行可分为对目标的任务的下派、目标应用的预处理、命令的生成、命令的执行、执行结果的处理。这个流程是对应用进行分析的比较通用的流程，所以我们的框架经过一定的修改是可以进行一定的功能扩展。

4.4.2 Drozer 及其插件的实现

在基于动态任务分配的并行分析系统 paraIntentFuzz 中采用 Drozer 给目标应用发送命令。Drozer 是由 mwrlabs 使用 python 开发的一款开源的漏洞挖掘框架。Drozer 由 2 部分组成：Drozer 主程序和安装到手机上的 agent.apk 代理，它通过给代理发命令，然后代理再发给目标的软件，最后又将执行结果返回给 Drozer 主程序。为了方便给 Drozer 发命令，我们采用 Drozer 的插件机制开发了对命令进行封装的插件，能够直接发送命令，而不用在命令行中登录 Drozer，这样极大地简化了发命令的工作量。Intent 命令的发送、处理流程图如下图所示：

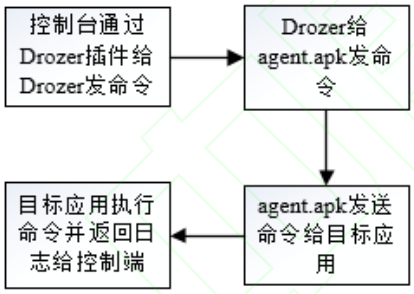

图 4 Intent 命令发送流程图

4.4.3 快照的使用

Android 为了让模拟器快速启动而设计了快照的功能，它就是将当前模拟器的状态，比如：连接网络的状态、应用程序打开后的设置等所有信息都保存下来，下一次从快照启动就可以很快地恢复到这一状态。我们把初始分析环境做成快照，在一批任务分析完成后，将模拟器以快照的方式重启，便可以免去对模拟器环境初始化等准备工作，大大提高了分析效率。

4.4.4 源码修改、日志的获得与处理

Android 中有多种方式对权限进行检查[20]，可以通过 PackageName、PID、Thread Status 等方式进行权限检查。对启动 Activity 的权限检查会通过 Android 源码 ActivityManagerService.java 中的 public int checkPermission(String permission, int pid, int uid)方法来进行检查，permission 是需要检查的权限名字，pid 是当前线程的父线程的 id，uid 是应用在系统内部标志编号。检查的权限结果返回值为 int，若通过则返回 0，不通过则返回-1。我们会将这个方法中的所有参数信息以及返回值都输出到日志中并加上特有的标记，最后通过我们的标记就可以对日志信息进行过滤，得到我们关注的日志信息。进行日志处理的时候只需要读取出权限的名字以及权限检查的结果就知道该组件是否有权限泄露。

5 实验

本章主要通过实验证明 paraIntentFuzz 框架的可行性以及加入 extra 信息所带来的效果。实验中 paraIntentFuzz 被部署到基于 Openstack 开发的平台，配置了 4 台虚拟机，配置均为双核、8GB 内存、20GB 硬盘，所使用的 Android 源码版本是 4.4.2，采用的是修改并编译好的镜像。

5.1 并行所带来性能提升

为了证明 paraIntnetFuzz 并行所带来的效率的提升，我们做了 2 组实验，分别是 4 台机器并行地去分析 100 个应用和 1 台机器去分析 400 个应用。四台机器并行分析的时候各台机器开始、结束的时间表如下表所示：

表 1　4 台机器分析的开始、结束时间

	第一台	第二台	第三台	第四台
开始时间	11:30:00	11:30:00	11:30:00	11:30:00
结束时间	14:27:06	14:23:43	14:22:03	14:21:08

从上面表格来看分别运行时间是：2 小时 57 分 6 秒、2 小时 53 分 43 秒、2 小时 52 分 3 秒、2 小时 51 分 8 秒，总共运行时间：11 小时 36 分 。

取上面表格中的开始时间的最小值和结束时间的最大值分别是：11:30:00 和 14:27:07 耗时是：2 小时 57 分 07 秒。

当使用一台机器去分析这 400 个应用的时候开始时间为：2016-7-24 16:00:40 和 2016-7-25 03:30:49，也就运行时间 11 小时 30 钟 9 秒，上面并行地运行，4 台机器用于分析的时间一共是：11 小时 36 钟，也就是说会比 1 台机器去分析要会多 5 分 51 秒，这可

能是有局域网内部传输任务所耗费的时间。但是这点时间开销也是可以接受的。从上面数据可以看出我们设计的 paraIntentFuzz 效果是很好的，效率大概是单机的 4 倍。

5.2 加入 extra 信息带来的效果的提升

为了证明加入 extra 信息之后能触发更多的权限检查，我们对 500 个随机来自国内的应用市场和 Google Play 市场的应用进行分析，在没有加入 extra 信息的情况和加入 extra 后的触发的权限检查以及通过权限检查的情况如下：

表 2 加入与未加入 extra 触发权检查次数

	触发的权限检查次数	触发权限检查且通过检查的次数
未加入 extra	3805	2390
加入 extra	5223	3581
提升比	37%	50%

通过上表可以看出加入 extra 后无论是触发的权限检查的次数还是通过权限检查的次数都有很大的提升，证明了加入 extra 的有效性。

5.3 动态的任务分配实验

为了证明动态地按大小排序进行分析是比较好的选择，我们对 400 个应用进行了 4 组实验，这 400 个应用全都随机来自国内的应用市场和 Google Play 市场，应用的大小从 1.3MB-80MB 不等，400 个应用总共 5.2GB。

5.3.1 初始时随机分配 100 个应用

刚开始分析的时候随机地为每一台机器分配 100 个应用，各台机器的分析运行时间如下表所示：

表 3 初始时随机分配 100 个应用运行时间

	开始时间	结束时间	耗时
第 1 台	07:57:50	16:13:59	8 小时 16 分 19 秒
第 2 台	07:57:50	16:24:07	8 小时 27 分 17 秒
第 3 台	07:57:50	17:45:28	9 小时 47 分 38 秒
第 4 台	07:57:50	16:19:21	8 小时 21 分 31 秒
最早开始/最晚结束	07:57:50	17:45:28	9 小时 47 分 38 秒

从上表可以看到第 3 台机器比第 4 台晚了 1 小时 26 分 7 秒完成对 100 个应用的分析。

5.3.2 初始时按预估的分析时间排序分配 100 个应用

在各运行结点运行之前首先将服务器端的任务队列中的 400 个应用按预估的分析时间递减排序并给每个任务进行编号 0-399,分 100 次传输，每次分别给 4 台机器各传输 1 个应用。设 n 取值范围为：[0, 100),k 代表机器的编号取值范围为[0, 3]，服务器结点编号为 0，其余 3 台编号 1-3，编号为 4*n+k 就给 k 号机器。各结点运行时间如下表所示：

表 4 初始时按预估分析时间排序分配 100 个应用运行时间

	开始时间	结束时间	耗时
第 1 台	10:36:00	19:28:46	8 小时 52 分 46 秒
第 2 台	10:36:00	19:18:32	8 小时 42 分 32 秒
第 3 台	10:36:00	19:08:51	8 小时 32 分 51 秒
第 4 台	10:36:00	18:52:51	8 小时 16 分 51 秒
最早开始/最晚结束	10:36:00	19:28:46	8 小时 52 分 36 秒

从上表我们可以看到第 1 台机器比第 4 台晚了 35 分 45 秒完成对 100 个应用的分析。

5.3.3 动态地分配，但不按预估的分析时间排序

服务器结点运行后获取任务队列的列表（此时不对列表中的任务进行分析时间的预估和排序，随机地分配给运行节点）。进行任务初始分配后就一直监听来自运行结点的任务请求，当每一个运行结点分析完一批后就给服务器发送一个任务请求，服务器再分配下一批任务给请求任务的运行节点。采用分析一批再分配一批的动态分配方式，各结点运行时间如下表所示：

表 5 动态地分配，但不按预估的分析时间排序运行时间

	开始时间	结束时间	耗时
第 1 台	09:44:03	18:40:12	8 小时 56 分 09 秒
第 2 台	09:44:03	18:16:39	8 小时 32 分 36 秒
第 3 台	09:44:03	18:29:55	8 小时 45 分 52 秒
第 4 台	09:44:03	18:22:29	8 小时 38 分 26 秒
最早开始/最晚结束	09:44:03	18:40:12	8 小时 56 分 09 秒

从上表我们可以看到第 1 台机器比第 2 台晚了 23 分钟 33 秒完成对 100 个的分析。

5.3.4 动态地分配，按预估的分析时间排序

和 5.3.3 的类似，只是获取任务队列的时候将所有任务按预估的分析时间多少降序排序，并先为每台机器分配预估会耗时较多的任务，最后分配耗时较少的任务，各结点运行时间如下表所示：

表 6 动态地分配，按预估的分析时间排序运行时间

	开始时间	结束时间	耗时

第 1 台	09:49:00	18:32:33	8 小时 43 分 33 秒
第 2 台	09:49:00	18:29:20	8 小时 40 分 20 秒
第 3 台	09:49:00	18:28:48	8 小时 39 分 48 秒
第 4 台	09:49:00	18:34:31	8 小时 45 分 31 秒
最早开始/最晚结束	09:49:00	18:34:31	8 小时 45 分 31 秒

从上表我们可以看到第 4 台机器比第 3 台晚了 5 分钟 47 秒完成对 100 个的分析。

5.3.5 四种调度策略效果对比

将上面的 4 中情况中 4 台机器中的耗时最大值统计如下表 8 所示：

表 7　以上 4 中种情况分析所需时间

任务分配策略	耗时最大值
初始时随机分配 100 个应用	9 小时 47 分 38 秒
初始时按预估分析时间排序分配 100 个应用	8 小时 52 分 36 秒
动态地分配，但是不按预估分析时间排序	8 小时 56 分 09 秒
动态地分配，按预估分析时间排序	8 小时 45 分 31 秒

由上表可知动态地分配，按预估分析时间排序这种策略就完成这 400 个应用的任务来说耗时是最少的。

5.3.6 动态地所带来的时间开销

通过实验发现从客户机发送连接到服务器端与其建立连接的时间在 2 秒内，传输待分析应用的速度可达 25MB 每秒，一个 50MB 的应用传输到客户端只需要 4 秒，相对于分析一个应用的时间来说这点时间是很少的。

5.4 大规模模糊测试实验结果

通过对 10064 个国内市场的和 Google Play 的应用进行分析得到如下的数据：

表 8　每种权限的泄露次数

权限名	暴露个数
android.permission.BLUETOOTH_ADMIN	1
android.permission.USE_SIP	1
android.permission.READ_PROFILE	2
android.permission.WRITE_CALENDAR	2
android.permission.DOWNLOAD_WITHOUT_NOTIFICATION	3
android.permission.WRITE_CONTACTS	3
com.airliners.android.permission.C2D_MESSAGE	3
android.permission.RECORD_AUDIO	4
android.permission.WRITE_SYNC_SETTINGS	3
android.permission.READ_SYNC_SETTINGS	4
android.permission.SET_WALLPAPER	4
android.permission.READ_CALENDAR	5
android.permission.EXPAND_STATUS_BAR	6
android.permission.GET_PACKAGE_SIZE	6
android.permission.SET_WALLPAPER_HINTS	6
android.permission.MANAGE_ACCOUNTS	8
android.permission.CHANGE_WIFI_MULTICAST_STATE	10
android.permission.WRITE_SYNC_SETTINGS	10
android.permission.KILL_BACKGROUND_PROCESSES	11
android.permission.WRITE_SMS	12
android.permission.READ_SYNC_SETTINGS	13
android.permission.CHANGE_NETWORK_STATE	16
android.permission.ACCESS_LOCATION_EXTRA_COMMANDS	20
android.permission.BLUETOOTH	27
android.permission.READ_CALL_LOG	29
android.permission.MODIFY_AUDIO_SETTINGS	30
android.permission.DISABLE_KEYGUARD	47
android.permission.BROADCAST_STICKY	54
android.permission.SYSTEM_ALERT_WINDOW	54
android.permission.CAMERA	56
android.permission.READ_SMS	58
android.permission.READ_CONTACTS	64
android.permission.AUTHENTICATE_ACCOUNTS	65
android.permission.RECEIVE_BOOT_COMPLETED	77
android.permission.READ_EXTERNAL_STORAGE	151
android.permission.GET_TASKS	455
android.permission.GET_ACCOUNTS	585
android.permission.WRITE_SETTINGS	707
android.permission.WRITE_EXTERNAL_STORAGE	838
android.permission.ACCESS_COARSE_LOCATION	882
android.permission.WAKE_LOCK	954
android.permission.CHANGE_WIFI_STATE	1361
android.permission.ACCESS_FINE_LOCATION	1733
android.permission.VIBRATE	1812
android.permission.ACCESS_WIFI_STATE	3307
android.permission.READ_PHONE_STATE	4470
android.permission.INTERNET	7324
android.permission.ACCESS_NETWORK_STATE	10960

由上表可知这 10064 个应用中共暴露了 48 种权限。通过上面的表格我们可以看出：有些权限是涉及很敏感的操作的，比如：android.permission.WRITE_CONTACTS 是允许写入联系人信息、android.permission.READ_SMS 是允许程序读取短信的内容。

不同的应用敏感权限暴露的次数是不一样多的，下表为应用暴露的权限次数：

表 9 应用暴露权限的次数

暴露敏感权限的次数	应用个数	暴露敏感权限的次数	应用个数
2	2527	15	33
3	1126	16	26
4	791	17	21
5	407	18	36
6	227	[19, 21]	46
7	109	[25, 30]	42
8	133	[31, 55]	77
9	67	[56, 100]	33
10	63	[101, 183]	15
11	26	430	1
12	63	668	1
13	25	788	1
14	46		

暴露次数最多的是 668 次和 788 次，分别是：美团和豌豆荚应用，其中：CHANGE_WIFI_STATE、ACCESS_NETWORK_STATE 、READ_PHONE_STATE 、 ACCESS_WIFI_STATE 、ACCESS_FINE_LOCATION、这几个敏感权限在这美团中暴露的次数最多；而 ACCESS_NETWORK_STATE 、 GET_TASKS 、ACCESS_WIFI_STATE 、CHANGE_WIFI_STATE 、ACCESS_FINE_LOCATION 、 BLUETOOTH 、READ_PHONE_STATE 这几个敏感权限在豌豆荚应用中暴露的次数最多。

不同应用暴露权限的种类数是不一样的，每个应用暴露的权限的种类数目统计表如下：

表 10 应用泄露权限种类数目表

泄露的权限种类数	应用的数目	泄露的权限种类数	应用的数目
1	1650	8	64
2	2809	9	37
3	1310	10	14
4	695	11	6
5	455	12	7
6	179	13	2
7	139		

从上面可以看出，绝大多数应用都是泄露的 1-4 种权限，少有的应用会泄露 10 种及以上的权限。

6 总结与展望

本文实现了通过静态解析 smali 信息获取组件通信的 extra 信息，通过动态任务调度实现了一款并行的权限泄露挖掘系统 paraIntentFuzz，完成了对 10064 个应用执行并行模糊测试的分析任务，耗时 216 小时 10 分，发现了 7367 个应用存在不同程度的权限泄露问题，共泄露权限 48 种。通过实验证明了并行所带来的效率的提升。

由于实验室可分配的机器有限，我们只对 4 台机器进行了并行实验，但是我们的系统完全可以扩展到更多台的并行之中。我们的 paraIntentFuzz 实现了对 Android 应用进行 Fuzz 的主要流程，接下来会扩展到对其他漏洞的挖掘之中。

参考文献

[1] 管理员.2015 年移动应用质量大数据报告.[EB/OL].(2016-04-08).[2016-8-15]http://bugly.qq.com/bbs/forum.php?mod=viewthread&tid=763

[2] Google.应用基础知识.[EB/OL].[2016-8-15] https://developer.android.com/guide/components/fundamentals.html?hl=zh-cn

[3] Sutton M, Greene A, Amini P.模糊测试强制性发觉安全漏洞的利器[M].段念，赵勇.第一版.北京：电子工业出版社,2013 年 10 月:17-27

[4] 沈金明. 基于系统日志的计算机网络用户行为取证分析系统的研究与实现[D]. 南京:东南大学,2006

[5] nccgroup. Intent Fuzzer.[EB/OL].(2012-6-6).[2016-8-15] https://www.nccgroup.trust/us/about-us/resources/Intent-fuzzer/

[6] MindMac.IntentFuzzer.[EB/OL].(2013).[2016-8-15] https://github.com/MindMac/IntentFuzzer

[7] Yang K, Zhuge J, Wang Y, et al. IntentFuzzer: detecting capability leaks of android applications. [C]//Proc of the 9th ACM Symp on Information, Computer and Communications Security.New York: ACM, 2014:531-536

[8] 王凯，刘奇旭，张玉清.基于 Fuzzing 的 Anroid 应用通信过程漏洞挖掘技术[J].中国科学院大学学报,2014,31(6): 827-835

[9] 安靖，杨义先，李献忠.路径条件驱动的混淆恶意代码检测[J].湖南大学学报:自然科学版，2013,40(9):86-90.

[10] 廖明华，郑力明.Android 安全机制分析与解决方案初探[J].科学技术与工程,2011,11(26): 6350-6635

[11] Google.Manifest permission.[EB/OL].[2016-8-15] https://developer.android.com/reference/android/Manifest.permission.html

[12] Google.Android Permissions.[EB/OL].[2016-8-15] https://developer.android.com/guide/topics/manifest/permission-element.html

[13] Google.应用基础知识.[EB/OL].[2016-8-15] https://developer.android.com/guide/components/fundamentals.html?hl=zh-cn

[14] Keith Makan, Scott Alexander-Bown.Android 安全攻防实战[M].崔孝晨，武晓音.第一版.北京：中国工信出版集团，电子工业出版社，2015 年 7 月：37-47

[15] Google.Intent.[EB/OL].[2016-8-15] https://developer.android.com/reference/android/content/Intent.html

[16] 拙_言. Android 五种数据传递方法汇总.[EB/OL].[2016-8-15] http://blog.csdn.net/xcl168/article/details/14436539

[17] Octeau D, Luchaup D, Dering M,et al.Composite Constant Propagation: Application to Android Inter-Component Communication Analysis.[C] // In Proceedings of the 37th International Conference on Software Engineering (ICSE), Florence, Italy.May 2015.

[18] 丰生强.Android 软件安全与逆向分析[M].第一版.北京：人民邮电出版社,2013 年 2 月：29-55

[19] MWR InfoSecurity.Drozer.[EB/OL].[2016-8-15] https://labs.mwrinfosecurity.com/tools/Drozer

[20] 闫梅，彭新光.基于 Android 安全机制的权限检测系统[J].计算机工程与设计, 2013, 34(3):854-858