这篇文章科普了Android应用发重打包方面的最新技术难点以及研究现状，针对我们之前的一份工作提出了改进，介绍了我们发表在ICSME2018的研究工作“RegionDroid: A Tool for Detecting Android Application Repackaging Based on Runtime UI Region Features”[1].

大家可以阅读我们上一份相关工作，“穿着马甲的重打包App,你在哪里？”。简单的阅读之后，大家对于重打包应用有初步的了解。重打包应用在生活中几乎无处不在，“xxx-破解版”的app让很多人免费用到了原本需要收费的应用或者功能，十分讨人喜欢，仿佛就是在做无私的奉献。这些应用就是重打包应用的一部分，第三方开发者对原来的应用进行反编译，然后根据自己的需求对应用进行添加、删除、修改部分功能，然后再重新编译发布成新的应用到市场上。所谓林子大了，什么鸟都有，这些人中可是有不少是带着坏心眼来做这件事的，我们之前的文章也提到，他们可能会做这些不好的事情：

* 插入额外广告获取经济利益；
* 插入恶意程序，窃取用户隐私信息；
* 植入或篡改链接，骗取流量或者诱使用户访问恶意网站；

这些心怀不轨的行为可真让人头疼，被这些骗子app坑的新闻也层出不穷。个人用户尤其是大爷大妈根本区分不出来，一不小心就被骗子利用。进一步说，现在国家版权保护加强，应用市场有责任也有义务对自己市场内的应用加强监管、监督，保障消费者的权益。

简单来说，就是有一大堆apk文件，如何快速有效的找出那些重打包的app呢？

**重打包检测现有方法**

要能够在实际应用场景中解决这个问题，自然是不能光靠人工审核的。目前有不少自动重打包检测相关的研究，这些研究中普遍有一个核心的思想就是给应用标注上一个独特的特征：胎记（birthmark）。顾名思义，就像人一样，每个人的胎记基本上独一无二（当然不排除可能高度相似的情况）。我们希望独立开发的应用能够有不同的胎记，重打包的应用原则上不能够修改应用的胎记，这样我们就能用这个胎记来对应用进行识别。

现阶段已经有很多工作来解决安卓重打包检测问题。按照app胎记的生成方式，我们将这些工作大致分为静态方式和动态方式这两类。静态方式大都是分析应用的反编译后的代码和资源，DroidMOSS [3]对应用中每个函数方法进行模糊哈希作为胎记； DNADroid [4]和AnDarwin [5]为建立了程序依赖图（program dependency graph），Centroid [6]构建了3D控制流图(3D control flow graph)，Viewdroid [7]则根据应用界面生成了界面图(view graph)，这些生成图的方法各自计算图的相似度从而决定应用的相似度。而在动态方式中，人们通过分析应用运行时的API调用序列[8]和UI信息 [9]来计算相似度。我们之前的工作Repdroid[2]也是动态方式的一种。

**Repdroid的困境：安卓开发技术的更新换代带来的冲突**

我们之前的工作Repdroid[2]提出了一种基于安卓应用运行时状态的软件胎记。我们动态执行安卓应用，可以获得若干个不同的界面以及不同界面之间的跳转关系，我们将相似的界面归为一个组，界面之间的跳转关系就可以转换成不同组之间的转换关系 。这样我们的胎记就是一个有向图（如下图），每一个节点是一个包含若干个相似界面的组，每一条边是组内部某个界面跳转到另一个组的某个界面的事件。我们将该图作为应用的胎记，然后比较app之间的胎记相似度来识别是否为重打包应用。

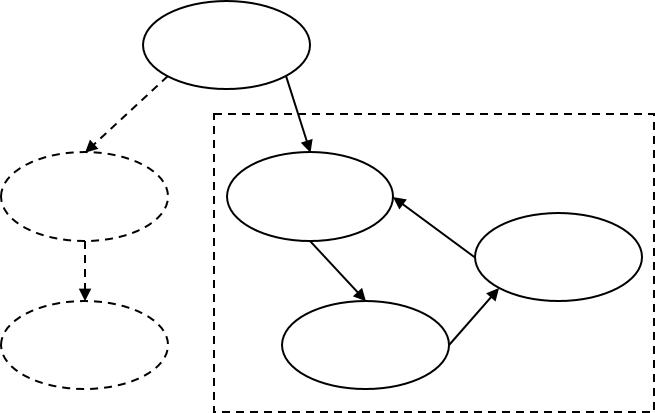
我们知道，应用的胎记是对应用的某些特征进行抽象化，胎记的优劣很大程度上依赖于对安卓应用的特征选择上。该胎记很大程度上依赖于对安卓应用界面的抽象化。然而，随着安卓开发技术的不断更新换代，市场上出现了一种新的技术潮流：混合应用。

混合应用是介于本地化应用和web应用两者之间的app。所谓本地化应用，就是正宗的原始安卓应用，例如你初学安卓时运用sdk和Android Studio写出的第一个HelloWorld应用。而Web应用依赖于html来与用于交互，可以理解成一个个网页，不需要用户安装，量级更轻开发更快。而混合应用集两家之长，它虽然看上去是一个本地化应用，但是使用的时候其实是访问一个浏览器，直白的说就是应用包装了一个客户端的壳，其实里面是html的网页。之所以混合应用越来越流行的一个原因是，市场变化越来越快，传统的开发效率较低，对于跨平台更是头疼，既要开发安卓版本的应用，又要开发ios的版本，对于小公司来说简直就是噩梦，而混合应用很好的解决了这个问题，相当于一套方案，跨多种平台。

混合应用给重打包带来的一个问题就是，安卓界面的表现形式与原生应用有很大的区别。原生安卓的界面可以表示成一个xml树，例如Button、Text就是这个xml树上的一个节点，Repdroid对界面的Encoding就是对xml树进行遍历，根据节点类型生成界面的一个向量表示。但是混合应用的界面中，节点类型几乎都是同的View，这样Repdroid就失效了，无法有效的表示一个界面。

**Repdroid生成胎记策略缺陷：随机游走的偶然性**

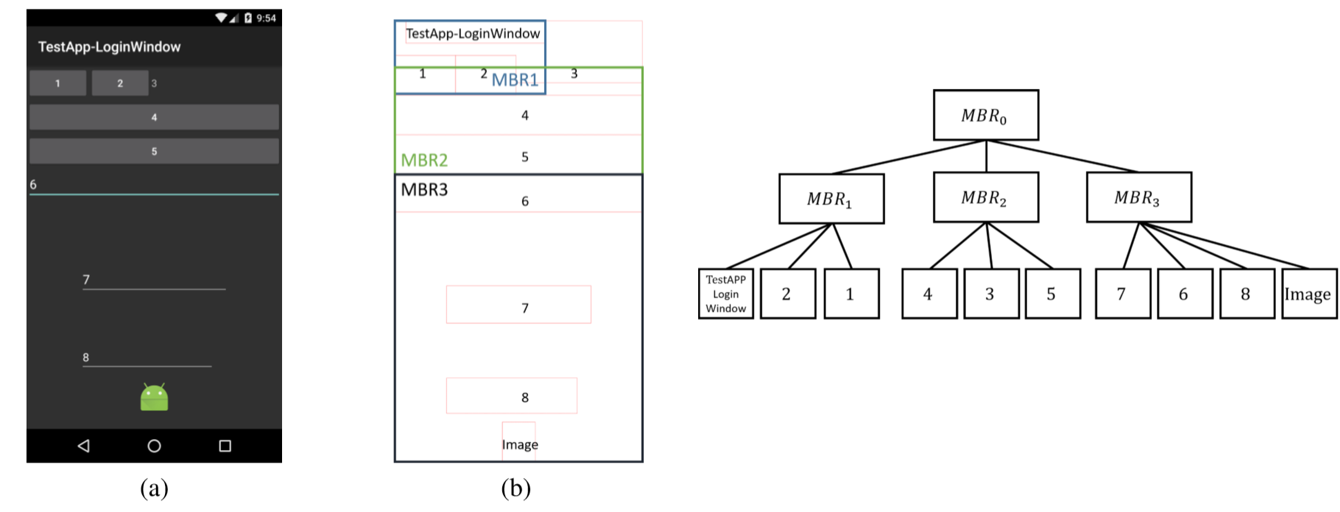
Repdorid还有一个问题源自于胎记生成的随机游走。胎记生成的策略简单来说，就是自动化的执行应用，给所有可以交互的区域可能执行的操作如点击、滑动、长按等等赋予相同的初始权重，然后带权重的随机选择每一步的执行操作，当操作之后产生了新的界面，我们就给这个操作更多的权重，这样可以倾向于触发那些能够产生新界面的操作。但是大家都知道，现在的安卓应用，有的已经功能丰富到难以想象，一个页面上有多个可以点击交互的区域，可以想象，我们如果随机的去触发某个操作，很难打开所有可能的界面，甚至会陷入某个区域，“沉迷其中难以自拔”，如下图。我们或许就在一个小范围内不断的循环，而对于其他部分的界面根本触发不到，这样我们即使对同一个应用也有可能生成不同的胎记。



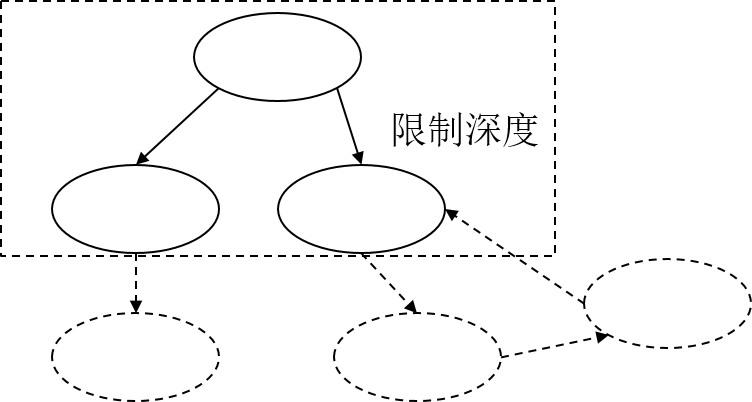
**我们的改进**

在Repdroid整体框架的基础上，针对Repdroid所存在的上述两个主要问题，我们提出了相应的改进方案。

解决第一个问题就是需要我们对每个界面进行更好的抽象表示。经过我们的观察，我们发现不管是原生应用还是混合应用，有一个不变的就是控件的空间区域特征。如下图，界面中每个控件都有一个矩形区域对应，这些矩形的位置极其大小构成了这个界面布局的一个骨架，对界面的部分修改不会对整体的骨架有太大的影响。我们用一个R-Tree来表示一个界面，这样就可以很好的处理混合应用带来的影响。

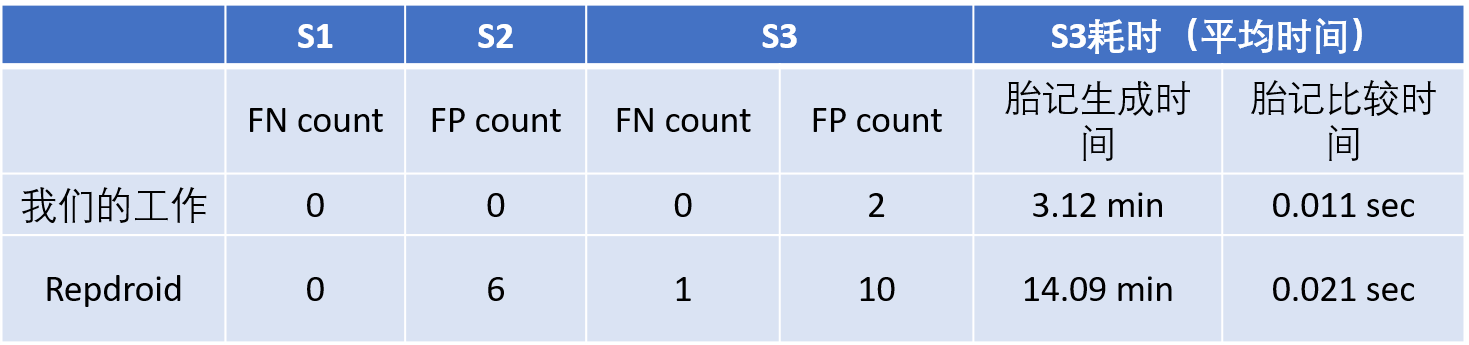


解决第二个问题的思想也很简单，这里有个比较简单的常识，人工测试者往往只需要观察一个应用初始的几个界面的特征，就可以判别应用之间是否是重打包。你可以想象一下，你打开两个类似的应用，是不是只要点击最开始的几个界面，你基本就能下判断，这不就是a抄了b吗。基于这个认识，我们对于Repdroid的胎记生成策略进行了改进。我们设置一个胎记图节点的深度范围，这个深度就是和启动界面的距离，当我们产生一个距离启动界面深度较深的界面，我们就向上回溯，将游走的深度限制。这样我们只需要获得那些相对启动界面比较靠近的界面，虽然不能获得完完整整的整个应用的所有界面信息，但是这已经足够我们生成胎记了。而且这也节省了不少的时间开销，真是又快又准。



**实验结果**

我们的实验包括两个部分，一部分和Repdroid一样，采用了相同的数据集S1和S2。S1共包含98对应用，每一对应用表示一个原始应用和一个被加密后的应用。S1数据集是用来检测我们的方法对加密应用是否有效。S2共包含从豌豆荚市场中8个分类下载的125个应用，该数据集是用来检测我们的方法能否正确的区分出不同的应用。第二部分是针对混合应用的实验，我们建立了数据集S3，共计157个应用，来自于AndroZoo以及另外7个混合应用框架开发如Appcelerator、Onsen UI等。最终的结果在下表中。表中FN表示事实上是一组重打包应用，但是方法并没有检测出来；FP表示事实上不是一组重打包应用，但是方法却判为重打包。



通过上面的实验，我们可以看到

* 对于被加密混淆的应用S1，我们的改进方法与Repdroid同样能检测出所有的重打包情况；对于S2里面拥有相同主题相同类别的独立开发应用（比如同样都是新闻阅读应用），我们同样有效的区分出它们，而且减少了误判。
* 对于混合应用，相比Repdroid，我们能够检测出重打包的应用，同时也大大减少了误判。
* 另外的，由于限制了生成胎记图的深度，我们在生成胎记的时间上也取得了很大的进步，大大节约了时间开销。

**参考文献**

[1] Yue S, Sun Q, Ma J, et al. RegionDroid: a tool for detecting Android application repackaging based on runtime UI region features. In Proceedings of the xxth International Conference on Software Maintenance and Evolution, 2018

[2] Yue S, Feng W, Ma J, et al. RepDroid: an automated tool for Android application repackaging detection. In Proceedings of the 25th International Conference on Program Comprehension, 2017: 132-142.

[3] Zhou W, Zhou Y, Jiang X, et al. Detecting repackaged smartphone applications in third-party android marketplaces. In Proceedings of the second ACM conference on Data and Application Security and Privacy, 2012: 317-326.

[4] Crussell J, Gibler C, Chen H. Attack of the Clones: Detecting Cloned Applications on Android Markets. In Proceedings of ESORICS. 2012.

[5] Crussell J, Gibler C, Chen H. Scalable semantics-based detection of similar Android applications. In Proceedings of ESORICS. 2013.

[6] Chen K, Liu P, Zhang Y. Achieving accuracy and scalability simultaneously in detecting application clones on Android markets. In Proceedings of the 36th International Conference on Software Engineering, 2014: 175-186.

[7] Zhang F, Huang H, Zhu S, et al. ViewDroid: Towards obfuscation-resilient mobile application repackaging detection. In Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks, ACM, 2014: 25-36.

[8] Kim D, Gokhale A, Ganapathy V, et al. Detecting plagiarized mobile apps using API birthmarks. Journal of Automated Software Engineering, 2016, 23(4): 591-618.

[9] Soh C, Tan H B K, Arnatovich Y L, et al. Detecting clones in android applications through analyzing user interfaces. In Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, 2015: 163-173.

**论文信息**：" RegionDroid: a tool for detecting Android application repackaging based on runtime UI region features "。

**作者简介**：本文作者是来自南京大学计算机软件研究所的[岳胜涛](http://link.zhihu.com/?target=https%3A//github.com/qqdown)、孙清伟、[马骏](http://link.zhihu.com/?target=http%3A//moon.nju.edu.cn/people/junma/)、[陶先平](http://link.zhihu.com/?target=http%3A//moon.nju.edu.cn/people/xianpingtao/)、[许畅](http://link.zhihu.com/?target=https%3A//cs.nju.edu.cn/changxu)、吕建。