Dynamic Android Application Repackaging Detection

汇报人：岳胜涛 MG1533079

指导老师：陶先平 马骏

Application Repackaging

- Crack software's payloads
- Embed advertisements
- Insert malware

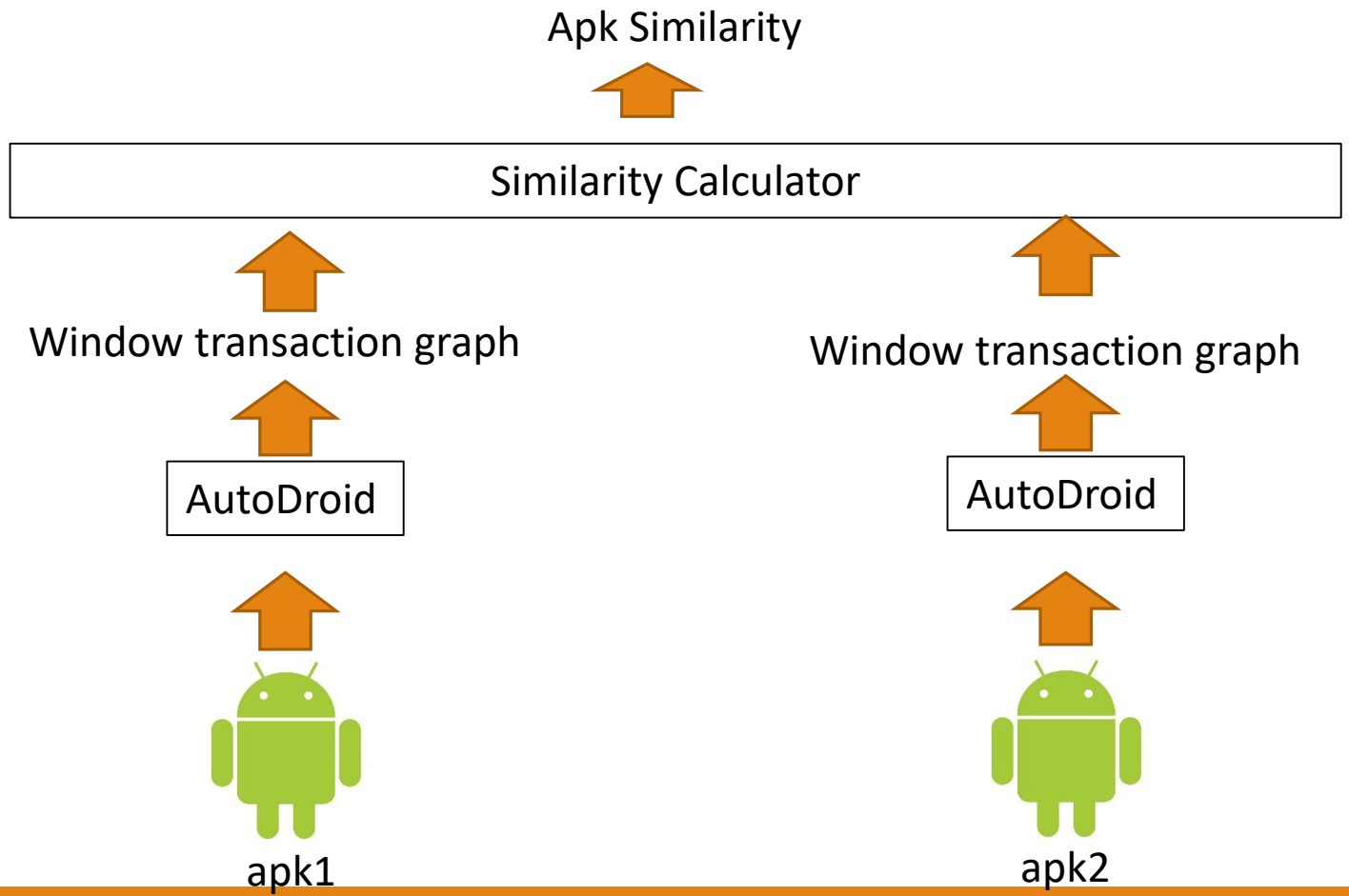
Application Repackaging

- The installation package (i.e. the .apk file) is easy to obtain
- Reverse engineering tools are readily available
  - Apktool, dex2jar, Baksmali/Smali and so on
- There is no certificate authorization
- Too many unofficial and third-party app markets exist
- 5% to 13% of apps are plagiarisms in the official Android market [1].
- 1083 of the analyzed 1260 malware samples (86.0%) are repackaged versions of legitimate apps with malicious payloads. [2]

Related Work

Static		Dynamic	
Code-level	UI-level	API-level	UI-level
Fuzzy Hashing[3]	View Graph[9] ResDroid[12]	API birthmarks[10]	Our works
Program Dependence Graph(PDG)[4,5]			
Feature Hashing[6]			
Module Decouple[7]			

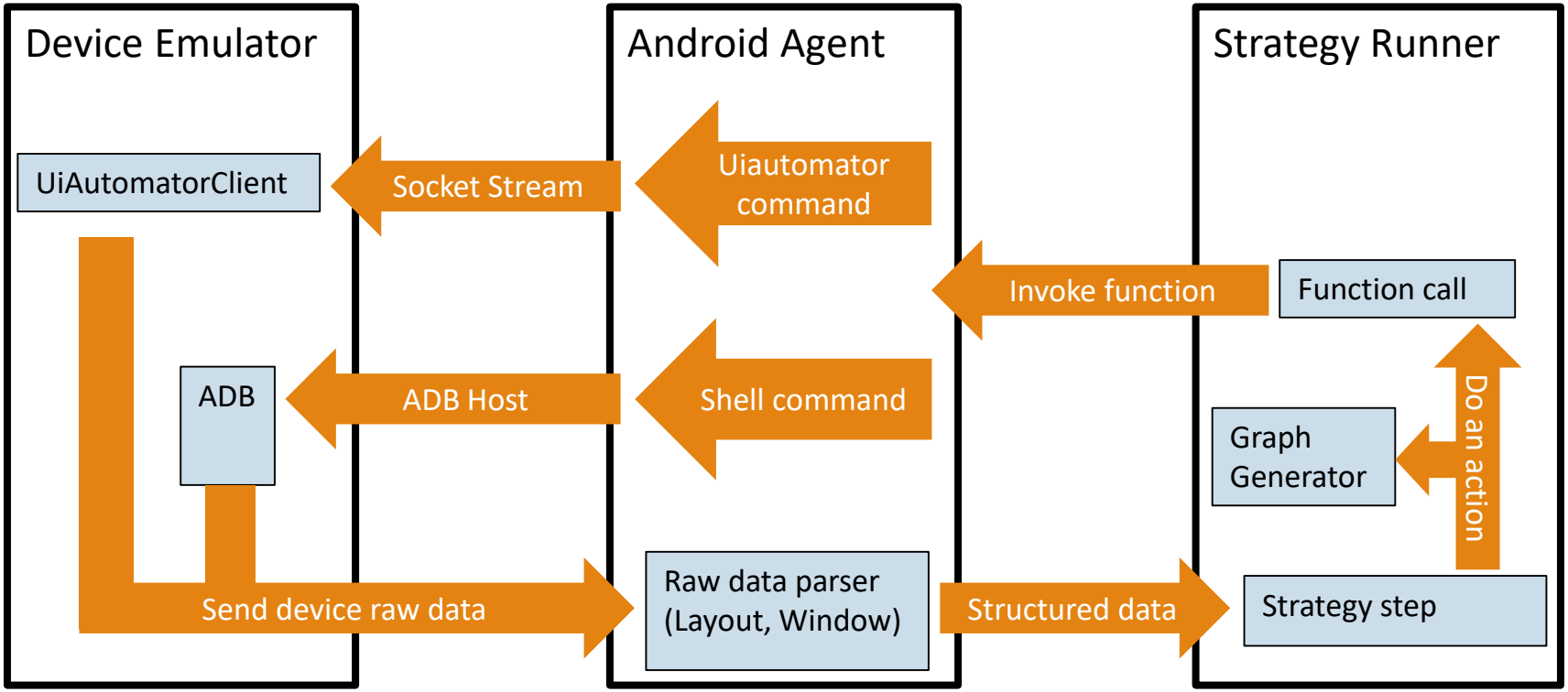
Our works


Apk Similarity
Similarity Calculator
Window transaction graph
Window transaction graph
AutoDroid
AutoDroid
apk1
apk2

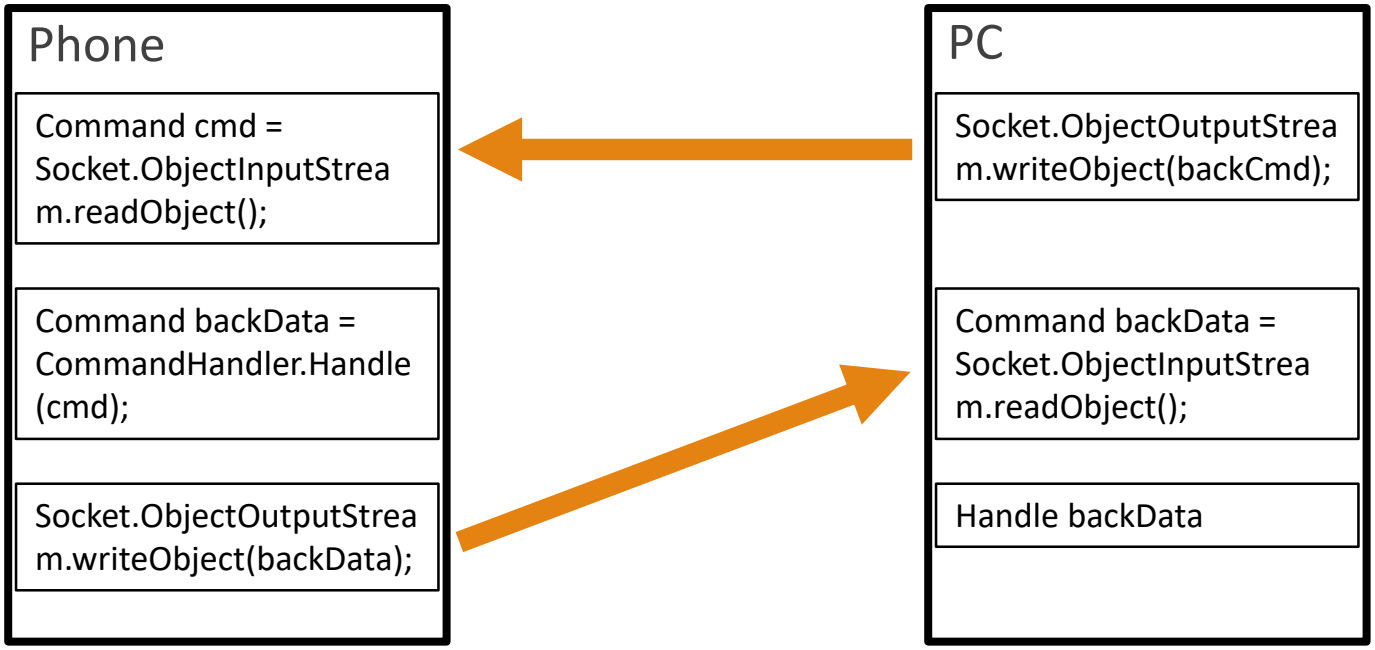
Our works

- Implement an automatic testing tool, AutoDroid, to generate the runtime Android application's window transaction graph.
- Propose a method to measure the similarity between two Android applications based on their window transaction graphs.

AutoDroid Structure



UiAutomator



AndroidAgent

IAndroidAgent	IAndroidAgent
boolean init()//*初始化*	**void** pressBack()
void terminate()//*终止操作*	String getLayout()
IDevice getDevice()//*当前设备*	String getRuntimePackage()
boolean installApk(String apkFilePath)	**boolean** doClick(LayoutNode btn)
String getFocusedActivity()	**boolean** doSetText(LayoutNode node, String content)
boolean startActivity(String activityName)	**boolean** doLongClick(LayoutNode node)
boolean stopApplication(String packageName)	**boolean** doClickAndWaitForWindow(LayoutNode node)
List<String> getRunningActivities()	**boolean** doScrollBackward(LayoutNode node, **int** steps)
List<AndroidWindow> getAndroidWindows()	**boolean** doScrollForward(LayoutNode node, **int** steps)
int getFocusedTaskId()	**boolean** doScrollToEnd(LayoutNode node, **int** maxSwipes, **int** steps)
void pressHome()	**boolean** doScrollToBeginning(LayoutNode node, **int** maxSwipes, **int** steps)

Strategy Runner

- Window Transaction Graph(WTG) [11]
  - A window transaction Graph of an Apk is a directed graph G(V,E), where V is a set of nodes, each of which represents a window. E is a set of edges < a,b > such that a ∈ V , b ∈ V and the smartphone display can switch from window a to window b by user actions.
- Window
  - A window is a logical data structure which contains the information about the screen.
- Layout
  - A Layout is dumped from a certain hierarchy view, which is in the form of XML.
  - LayoutTree is a logical tree created by the Layout XML.

Strategy Runner

- Depth(board) First Search strategy
  - Trying to traversal the whole application
  - However, almost impossible for most applications because the former states are hardly restored.
- Random strategy with exact action
  - Knowing the layout of the window, we can do the exact action on each layout node. As a result, it performs more efficiently than Monkey does.
- Bias weighted Random selection strategy
  - Random selection with bias weight.

Challenges

- According to the methods above, there are several challenges during our work and experiments.

Challenges 1

- WTG节点Window的确定
  - 以前的工作都是以Activity直接作为window的Identification，如果当前界面存在AlertDialog, PopupWindow或者Menu，那么本不应该相同的界面则被认为是同一个界面。因此我们决定使用当前界面的layout和windowID结合作为window的关键信息。

Challenges 2

- Algorithm 2: similarityWith(layoutTree1, layoutTree2)

  editDis = *EditDistance*(*layoutTree1.getTreeBFSHashes*(),

  layoutTree2.getTreeBFSHashes());

  return 1.0-editDis*/getTotalChildrenCount*();

  //*EditDistance* is a generics function, where the array element is in type of T.

  //所有节点(即使是自定义控件)的className都是该节点的基类名(Button, TextView等)

return hash;

Challenges 3

- 遍历策略

Algorithm 3: SelectionStrategy

currentWindow = *getCurrentWindow*()

while(currentWIndow != null)

    action = *getNextAction(currentWindow*, lastAction);

    switch(action)

        case NoMoreAction:

        return true;

        case NoAction:

        break;

        default:

        DoAction();LogAction();

        currentWindow = *getCurrentWindow*();

        WTG.addTransaction(fromWindow, toWindow, action);

    endswitch

endwhile

Challenges 3

- 遍历策略

Algorithm 4: WindowWeightSelectionStrategy : SelectionStrategy
@Override *getNextAction(currentWindow*, lastAction)
if WTG doesn't change any edge or vertex aftar doing lastAction
    lastAction.weight -=1;
    if(lastAction.weight < 1) lastAction.weight=1;
    nochangeCount++;
else
    lastAction.weight += 2+*averageWeight*(currentWindow.layout);
endif
nodeList = layout.nodes && node can be interacted
if nochangeCount >=threshold **return** NoMoreAction;
action.node = *randomSelectBasedOnWeight(actionList*);
action.type = randomSelect(actionList);
return action;

带权重的随机选择

初始状态时，所有节点权重为**1**

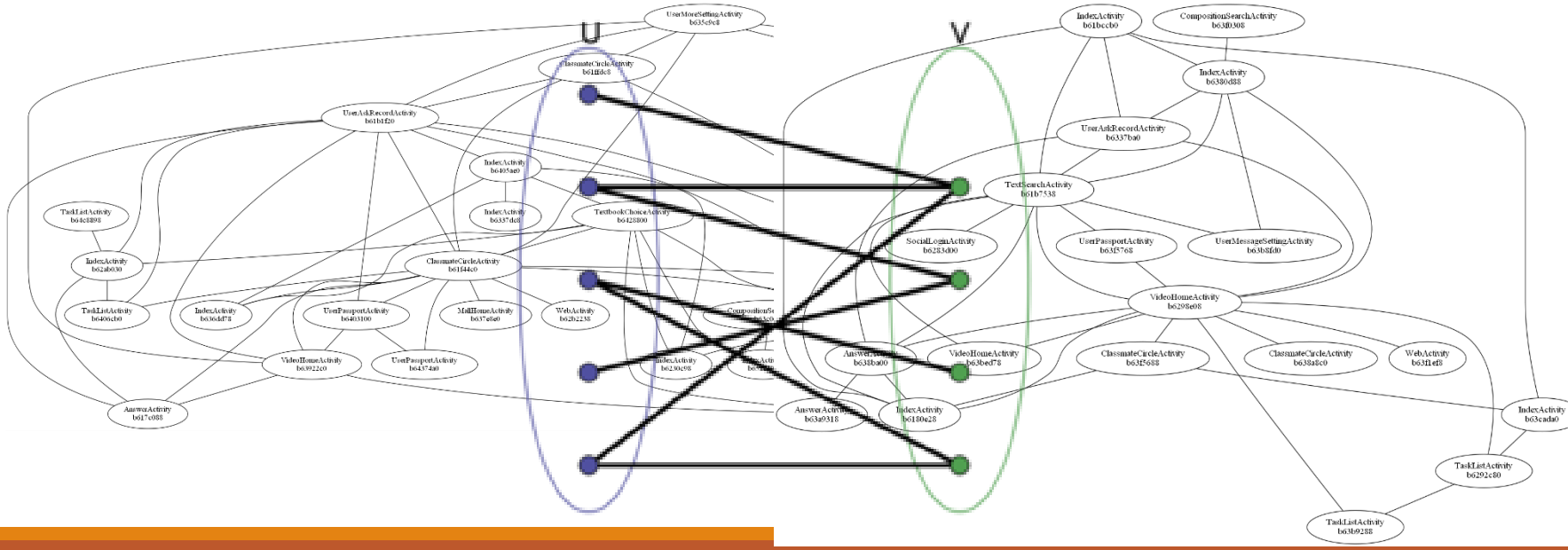
lastAction操作之后，WTG的**没有发生变化**，则该Action对应的节点权重将被**减少**

否则，权重**增加**，如果跳至了新的窗口，会把新窗口中所有节点的**平均权重**也加到这个节点上

特殊处理：会将返回、菜单按钮一并加入计算

Challenges 4

- Graph相似度计算:带权值的二分图匹配(KM算法)
  - 二分图由2个WTG的节点两两配对组成
  - 权值为两个节点相似度，同样利用之前的Window相似度算法作为节点的相似度，相似度小于一个阈值时，权值设为0
  - 计算得到最大的匹配权值W, sim = W/min(G1.vetexCount,G2.vetexCount)



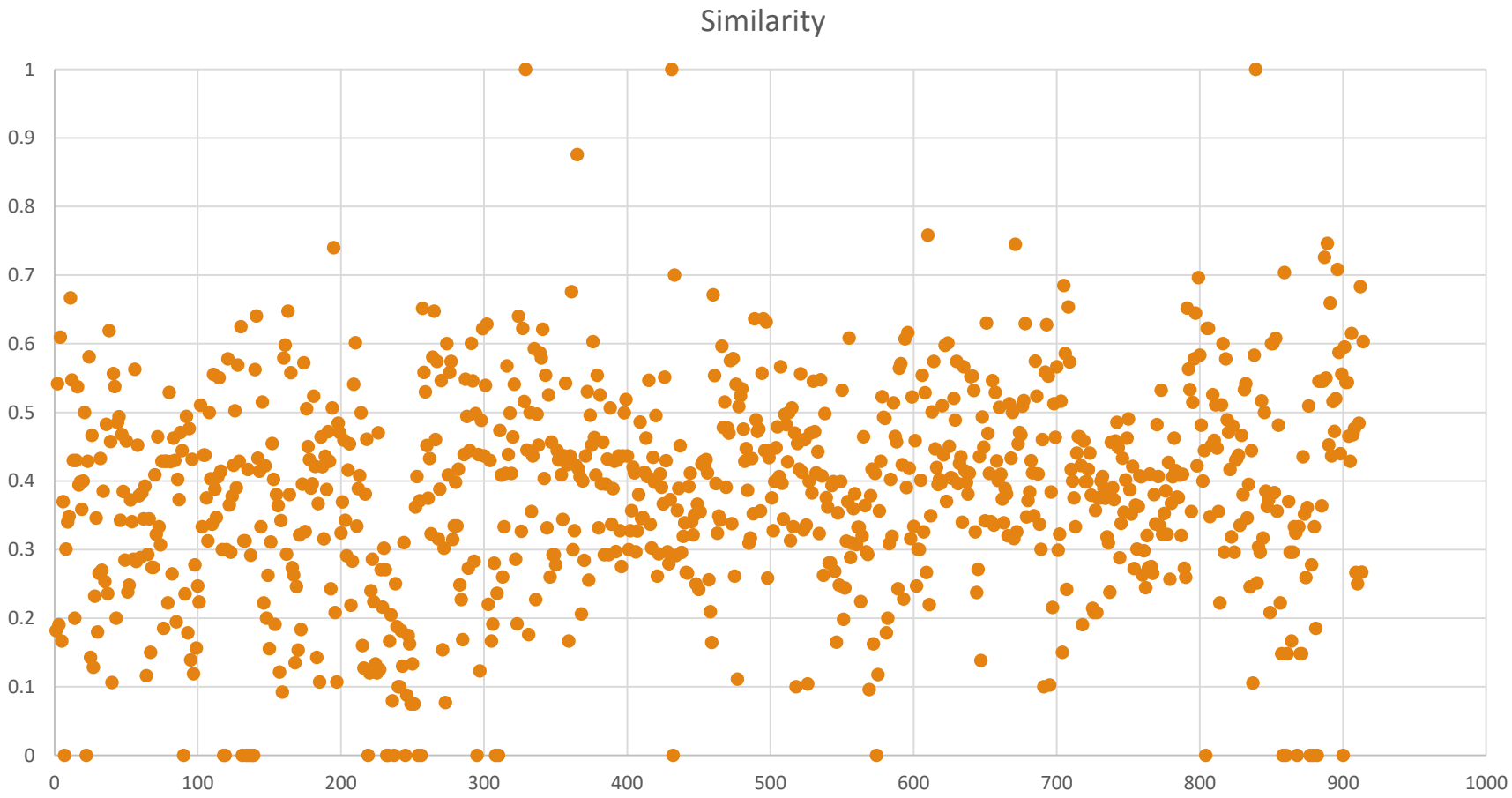
Experiments

- Resistance
  - 一个应用与其一个（通过程序转换技术获得的）副本应该被检测出相似
  - 利用爱加密来进行实验
    - APK中的classes.dex被加密，无法被反编译
    - APK被优化，API函数调用被修改
  - 共计15个APK被加密，被检测出15组相似，FN_Rate = 0%, FP_Rate = 0%
- Discrimination
  - 实现同一规约的独立开发的不同应用的应该认为是不同的应用.
  - 挑选了应用市场中的5个分类，每个分类挑选若干个应用进行实验
  - 共计5个分类87个APK，被检测出5组相似，经过人为判断，其中4组为确实为同一款应用（版本、部分外观不同），1组为误判，FP_Rate = 1.1%。该误判是由于该组Window个数过少，导致最后的相似度值过高。

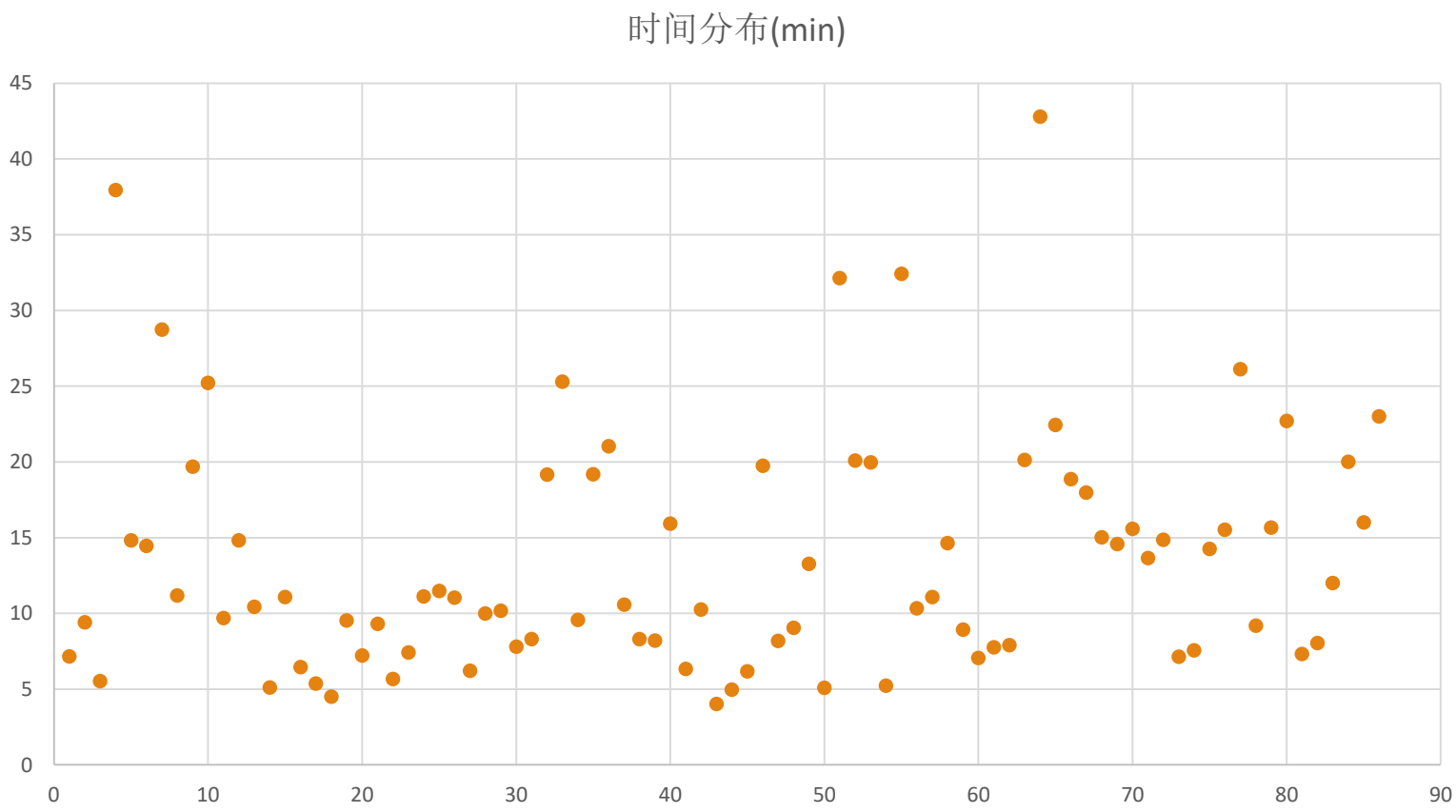
Experiments

apk1	apk2	Graph similarity	Vertex count 1	Vertex count 2	Vetex sim	Final sim
0f549bf753685d02049f0f87750fbdbc	0f549bf753685d02049f0f87750fbdbc	4	6	4	4	1
0f549bf753685d02049f0f87750fbdbc_new0	0f549bf753685d02049f0f87750fbdbc_new0	4	5	4	4	1
0f7eabb88512189109d1cfad6ba44951	0f7eabb88512189109d1cfad6ba44951	3.923809524	9	5	5	0.784762
1b483ad34282a0822db64d3821dfa738	1b483ad34282a0822db64d3821dfa738	7.5	9	8	8	0.9375
1e3793bc428c823d883f2bec95289144	1e3793bc428c823d883f2bec95289144	5	7	5	5	1
38dd5a548134514aa281a64a3349acff	38dd5a548134514aa281a64a3349acff	3	8	3	3	1
47914f8549fbc29eadddc79f36892020	47914f8549fbc29eadddc79f36892020	7.385336743	12	8	8	0.923167
49ac5a364159a7de007d501f97987912	49ac5a364159a7de007d501f97987912	4	5	4	4	1
5933a8cf7d55991921c04be51327f532	5933a8cf7d55991921c04be51327f532	2	2	2	2	1
618e186bf447356128e70f1b21a206c0	618e186bf447356128e70f1b21a206c0	6.256493506	9	8	7	0.782062
69f2319c6a9606fe30fcecb2a1b531db	69f2319c6a9606fe30fcecb2a1b531db	4	7	4	4	1
7463372e633e7905c1a40c6ac7604374	7463372e633e7905c1a40c6ac7604374	3	3	3	3	1
756809df9f533cae5476b5825c6d9ccb	756809df9f533cae5476b5825c6d9ccb	4	4	4	4	1
92cc1d557f7cd56a7e5c4643c1590fce	92cc1d557f7cd56a7e5c4643c1590fce	3	3	3	3	1
ahsan.my.lytish.1409802619785	ahsan.my.lytish.1409802619785	1	5	1	1	1
b14339f142c3b846becb983400f8707b	b14339f142c3b846becb983400f8707b	3	5	3	3	1

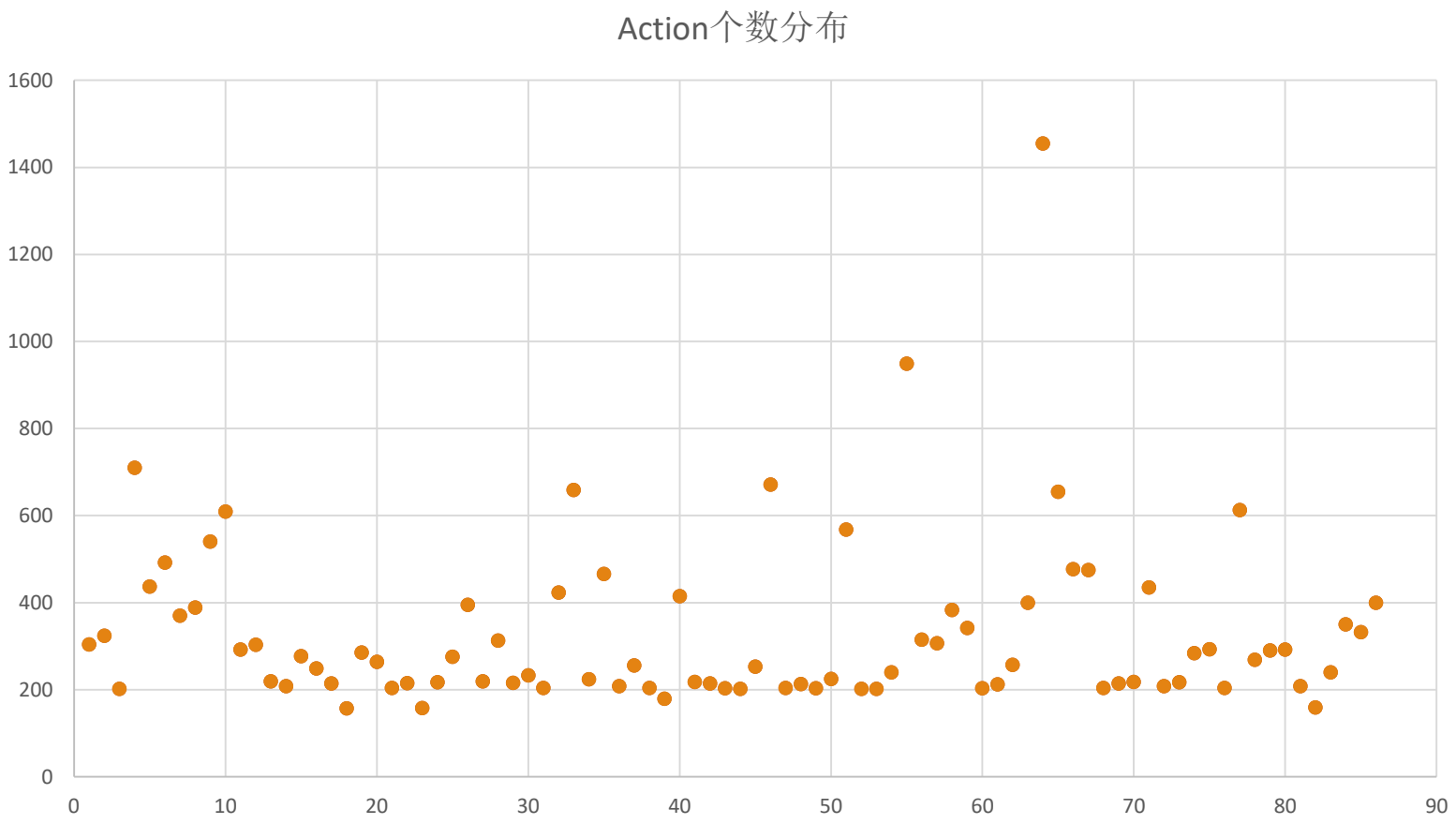
Experiments



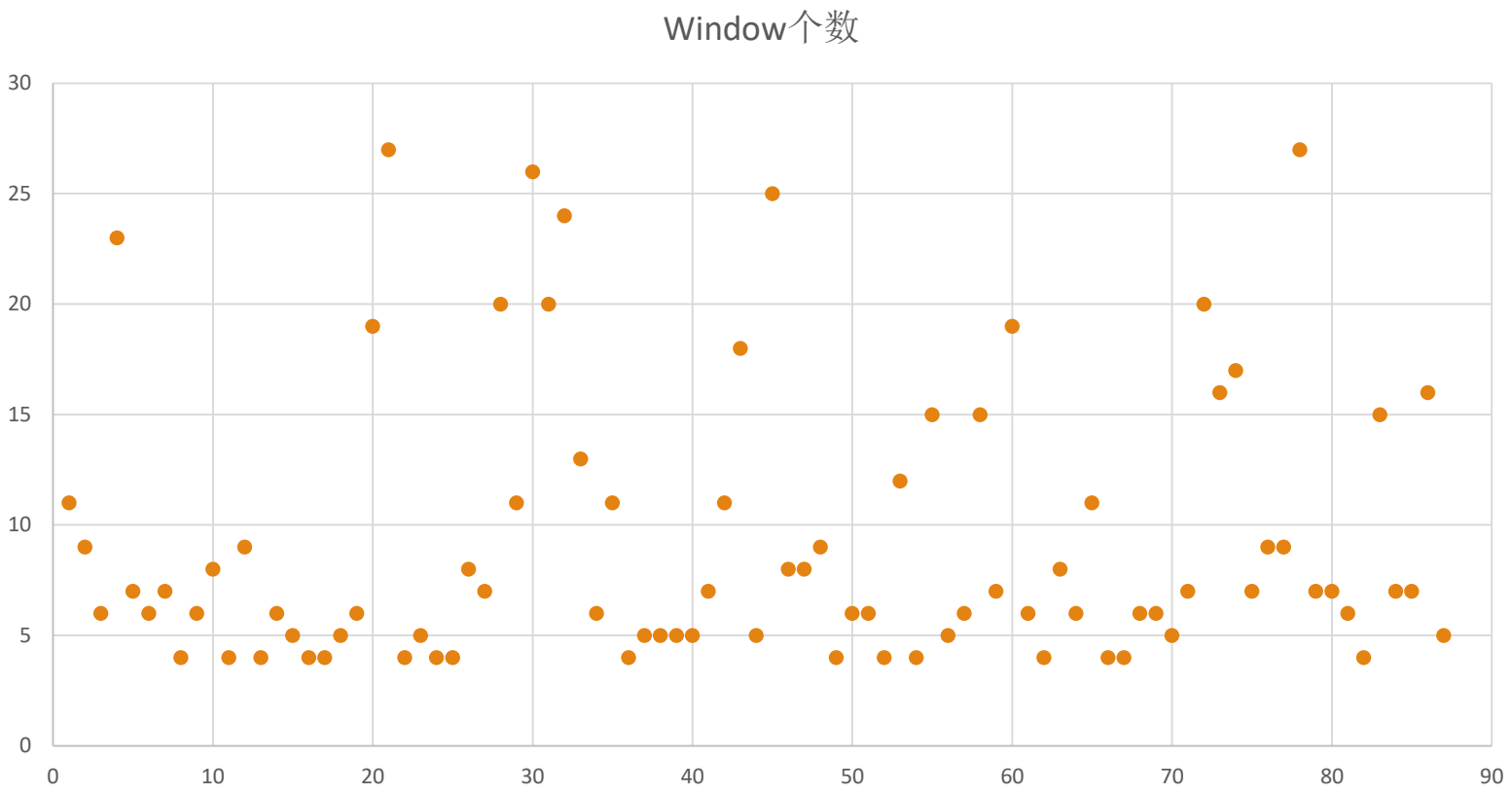
Experiments



Experiments



Experiments



Limitations

- 无法处理需要特定输入或者行为的情况
- 本工作以UI为核心，当应用UI数量较少时，在相似度判别上会出现错误。
- 本工作无法处理混合应用，这类应用采用HTML+JavaScript+CSS等Web技术快速架构应用的界面，因此利用我们的方法无法解析当前界面的Layout信息。

Reference

[1]Zhou, W., Zhou, Y., Jiang, X., Ning, P.: Detecting repackaged smartphone applications in third-party android marketplaces. In: Proceedings of the 2nd ACM Conference on Data and Application Security and Privacy, pp. 317–326. ACM (2012)

[2]Symantec Inc. Android threats getting steamy (May 7 (2011), http://www.symantec.com/connect/blogs/android-threats-getting-steamy

[3] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party Android marketplaces. In Proceedings of the second ACM conference on Data and Application Security and Privacy, 2012.

[4] J. Crussell, C. Gibler, and H. Chen. Attack of the clones: Detecting cloned applications on android markets. In ESORICS, pages 37–54, 2012.

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

[6] S. Hanna, L. Huang, E. Wu, S. Li, C. Chen, and D. Song. Juxtapp: A scalable system for detecting code reuse among android applications. In Proceedings of the 9th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, 2012.

Reference

[7] W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou. Fast, scalable detection of piggybacked mobile applications. In Proceedings of the third ACM conference on Data and application security and privacy, pages 185–196. ACM, 2013.

[8] A. Desnos and G. Gueguen. Android: From reversing to decompilation. In Black hat 2011, Abu Dhabi.

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

[10] Kim D, Gokhale A, Ganapathy V, et al. Detecting plagiarized mobile apps using API birthmarks[J]. Automated Software Engineering, 2015: 1-28.

[11] Yang S, Zhang H, Wu H, et al. Static Window Transition Graphs for Android (T)[C]//Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on. IEEE, 2015: 658-668.

[12] Shao Y, Luo X, Qian C, et al. Towards a scalable resource-driven approach for detecting repackaged android applications[C]//Proceedings of the 30th Annual Computer Security Applications Conference. ACM, 2014: 56-65.

Thanks for watching!