العنوان:	Detection Of Redestributed Malware Behavior In Android App Versions
المؤلف الرئيسي:	Al Salehi, Alaa
مؤلفين آخرين:	Abu Samra, Aiman Ahmed(Advisor)
التاريخ الميلادي:	2015
موقع:	غزة
الصفحات:	1 - 89
رقم MD:	768547
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	الجامعة الإسلامية (غزة)
الكلية:	كلية الهندسة
الدولة:	فلسطين
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسوب، البرمجيات الخبيثة، برنامج أندرويد
رابط:	https://search.mandumah.com/Record/768547

1- Introduction

1.1 Topic Area

Smart phones are becoming more integrated and important part of people's daily lives due to their highly powerful computational capabilities, such as email applications, online banking and online shopping…etc.

Malware, short for malicious software, is one of the major security threats in information systems. Malware includes viruses, worms, Trojan horses, spyware, dishonest adware, most root kits, and other malicious and unwanted software [1].

Android is an OS for smart phone owned by Google Inc, Google wants Android to become dominant in smart phone field, so they create their market to be an open market for developers with easy conditions for publishing new apps. In addition, Google opened Android for company solutions –companies can deploy their own modification on Android OS, Also Google allows Android`s users to install apps from other markets –there is a lot of android markets like Amazon store, SildeMe, Aptoide,…etc - and even form a website –unknown source-.This makes android a great environment for developers, marketers, users and companies.

This tremendous increase unfortunately, also makes android target for Malware applications and application`s thieves. Malware applications become the main threat field because of large custom and private data can be collected form user smart phones like Identifiers Disclosure - individually phone number, International

Mobile Equipment Identity number (lMEI)-, SMS, call log , contacts, browser history, location and emails. In addition, Malware can misuse SMS for Premium messages and root exploits. [2, 3, 4, 5]

In addition to malware android is a hot business field for developers also repackage app can threat their businesses. There are several ways developers may lose potential revenue: a paid application may be “cracked” and released for free, a free application may be copied and re-released on other markets with changes to the ad libraries or even in the same market with changes on interface and services. That will cause ad revenue or paid price goes to the plagiarist.

1.2 Research Question

The popularity and adoption of Smart phones has greatly stimulated the spread of mobile malware, especially on the popular platforms such as Android. In light of their rapid growth, there is a pressing need to develop effective solutions. General countermeasures to Android malwares are currently limited to signature-based antivirus scanners, which efficiently detect known malwares, but they have serious shortcomings with repackaged, refectories and redistributed. These maybe on threads, on versions, on components or maybe on different applications.

So the question is how to detect these behaviors on apps?

1.3 Significance

Tremendous increase of android markets make it easy for anyone to publish apps and update these apps. There is also a rising danger associated with Malware applications at mobile devices, so the problem of detecting Malwares is an interesting topic. In fact, 86% of detected malwares are old malware repackaged in new apps [6]. However, the fact all antimalware and antivirus focus on the current app version and they do not count malwares distributed on different versions of the same application.

In this research, we introduce a way to detect distributed malware on app version application and propose new way to analyze application against redistributed malware.

1.4 Thesis Structure

This thesis is organized as follows:

Chapter 1; Introduction: In this chapter thesis provides an introduction about thesis problem, questions and significance, this chapter describes why we choose this title for thesis and the idea of proposed solution.

Chapter 2; Background and Related Work: This chapter provides a background about Android system, application and programming. It also talks about malwares in general and malware in Android applications, at the end of this chapter there is a group of related work in the same topic of this thesis.

Chapter 3; Research Approach and Tools: This chapter describes in theoretical view the most important used tools in this thesis; it provides readers with description about used applications.

Chapter 4; Attack model: This chapter describes the model assumed on attack and the idea of distribution malware behavior.

Chapter 5; Methodology Evaluation and Analysis. Here readers can show the used methodology for thesis, and how we prove the feasibility of our idea, details of DMDA algorithm, this chapter also provides details about experiments and it results, in addition, it provides more details about algorithm.

Chapter 6; Conclusion and Future Work: A complete conclusion has been written in this chapter; also, we talked about future work related to his topic.

References: this chapter is a list of all sources associated with thesis.

Appendices: In this chapter, author attaches sample on the attack thread and code implementation for the proposed algorithm.

العنوان:	Detection Of Redestributed Malware Behavior In Android App Versions
المؤلف الرئيسي:	Al Salehi, Alaa
مؤلفين آخرين:	Abu Samra, Aiman Ahmed(Advisor)
التاريخ الميلادي:	2015
موقع:	غزة
الصفحات:	1 - 89
رقم MD:	768547
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	الجامعة الإسلامية (غزة)
الكلية:	كلية الهندسة
الدولة:	فلسطين
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسوب، البرمجيات الخبيثة، برنامج أندرويد
رابط:	https://search.mandumah.com/Record/768547

2- Background and Related work

2.1 ANDROID BACKGROUND

Android is a modern mobile platform that is designed to be truly open platform. Android developers use advanced hardware and software, as well as local and remote data, exposed through the platform to bring innovation and value applications to consumers.

2.1.1 Android System Architecture

The architecture of Android is implemented as a software stack, customized for mobile devices. Figure 1 Android some of the most important components of this stack [7].

The core of the Android platform is a Linux kernel. The kernel is responsible for handling device drivers, resource access, memory process, power management and other typical OS duties. The kernel also acts as an abstraction layer between the hardware and other software stack.

On top of the kernel are several native C/C++ libraries and Dlaivk VM. On the top of this layer there is Application framework of android which is responsible of managing android component lifecycle and interaction between android applications and low level APIs like media framework, OpenGL and etc.. On top of application framework there is application layer which contains contact, phone, SMS and E-mail applications.

2.1.2 Android Application Entry points

Android provides a Software Developer Kit (SDK) to developers. This SDK exposes the API needed by developers to build applications. Unlike java application, that has one entry point for application –main method- and works on one program architecture, android application has multi-entry point and works on message passing architecture. These multi entry points are:

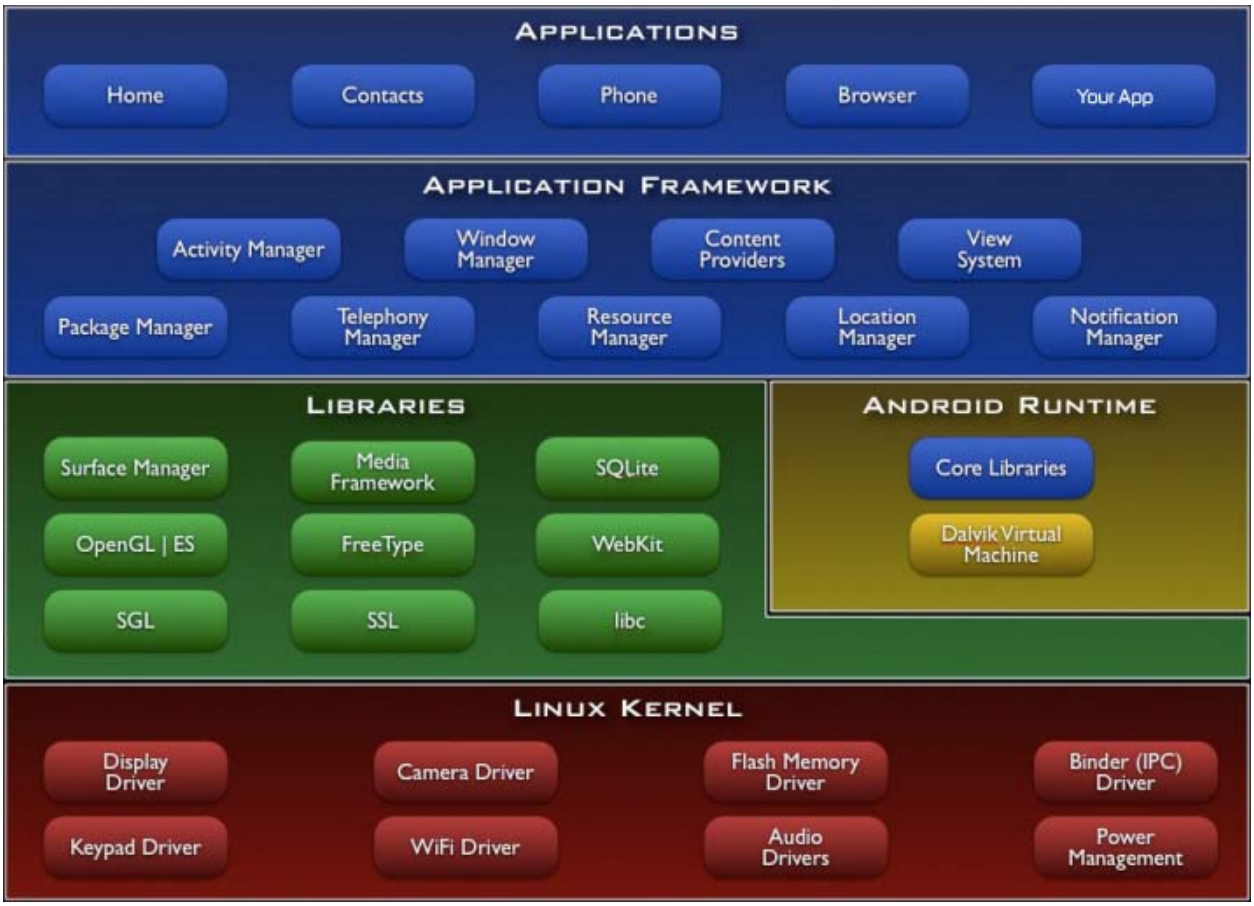

Figure 1 Android Architecture (Source Android developers) [8]

A. Activities

In Android, an activity is an essential components of an application. Every visual application should at least have one activity, the “main” activity even Launcher app-special application which is running when android user open their device- contains activity. Usually, an application has many activities and these can start each other, where each activity holds its state (e.g. start, stop or pause). An activity is the component that provides graphical user interface (GUI) for users.

Activity is one of the most complex and important component in android, android framework focuses on making activity optimized for users –by optimized battery and memory usage- and fixable for developers –by lifecycle callback- as much as possible. Figure 2: Activity Lifecycle explain how android framework manage activity lifecycle, onCreate and onDestroy called when activity start lives in memory and when activity ready to remove from memory. Onstart and OnStop called when activity start showed to user and hidden from user. OnResume and OnPause called when activity gain and lose focus of user. OnRestart called when activity started after stopped. Those are the main callback in activity lifecycle and there is others but less important [9].

B. Services

Services are components in Android that do not provide any user interface, and always run in the background to process long running operations. Services are

starting by other components –even in other applications- , so other component as activity or service can start a service. Android defines two types of Service bounded and unbounded. Bounded service life is independent on the component that started it. Unbounded Service does not depend on any component and any component can starting or stopping it.

C. Broadcast Receiver

Broadcast receiver is a mechanism that defines how Android operating system forwards its events to applications. The main usage of these broadcast receivers is inter-process communication and tracking of specific events (e.g. arrival of an SMS). Applications declare statically or dynamically their interest in receiving a certain event and accordingly the OS will try to deliver this event when it happens. Android defines two types of Broadcast Receivers: ordered and normal.

The normal Broadcast receiver is asynchronous and there is no order according it, which applications registered to get a broadcast, would receive the event first. As for the ordered ones, a priority can be set to require from the system to deliver the event to each app in a certain sequence, and some apps will get the event before others. This feature allows developers to capture and possibly modify the event`s carried data before it reaches to lower-priority consumers. In order case, an app can prevent other apps from getting specific event by aborting the

received data. Broadcast on android is one type of messaging on message passing architecture where sender send message to group of receivers.

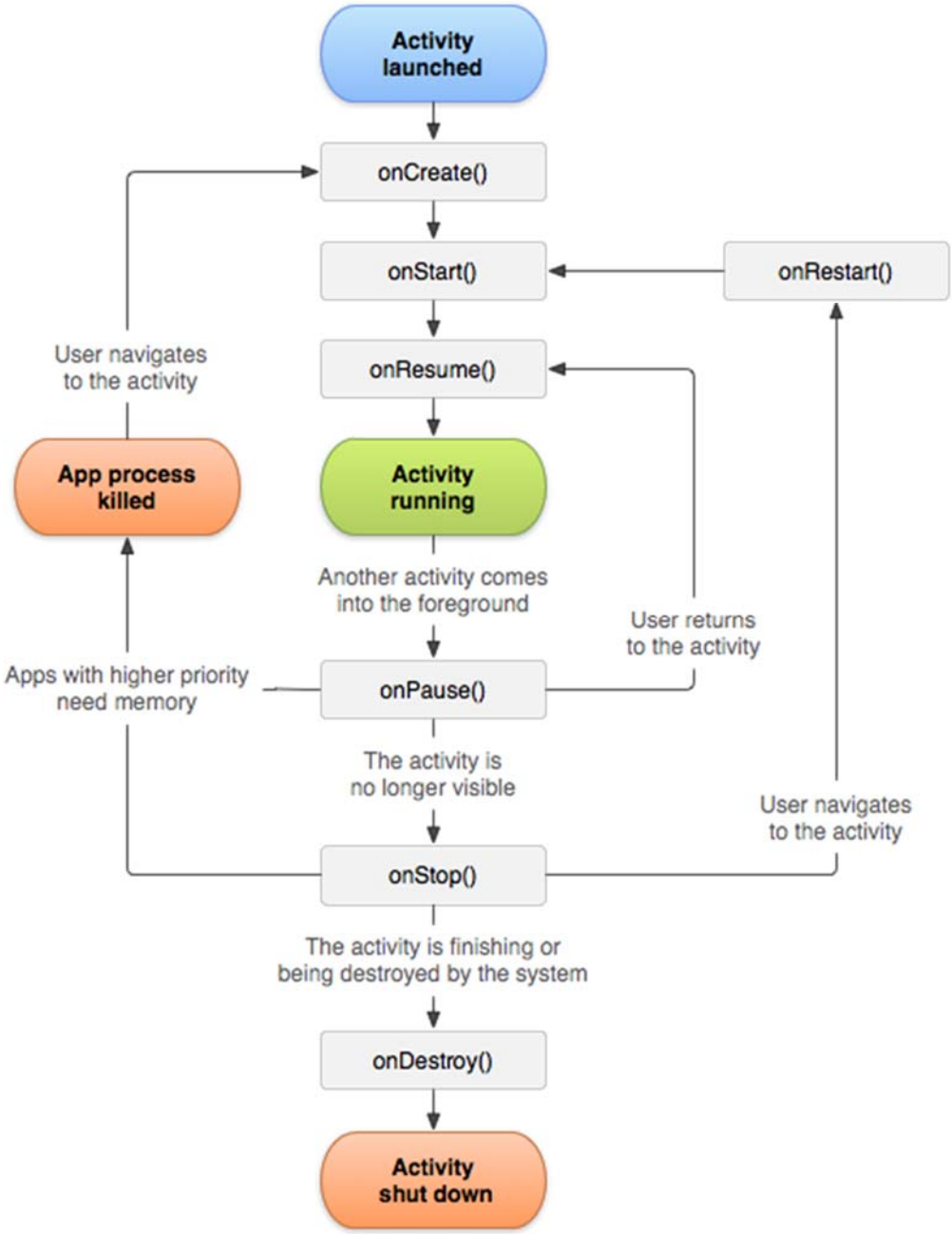

Figure 2: Activity Lifecycle (Source Android developers) [10]

2.1.3 Android Application Structure

Applications in android platform distributed in files called Android Packages (APKs). These files contain everything that the application needs to run from resources like images and XML files specifying UI layouts to the application code and metadata about what is the component of the application. APKs also include a manifest XML that specifies a number of metadata about the application, including its name, version information, the package (or namespace) of the code, the permissions it requires to execute, the component it contains and much more. Android applications are primarily developed in Java, sometimes native code may be used. The Java source code compiled to Java byte code and then converted into the Dalvik executable (DEX) format. Although similar to Java byte code, DEX byte code is incompatible with the Java virtual machine and instead runs on the Dalvik virtual machine. The conversion of Java byte code to DEX byte code is easily reversible and there are several tools can handle it.

2.1.4 Delvik VM

Android allows developers to run their application on top of virtual machine –known as Delvik VM-. Delvik VM created to handle limited memory size –about 60 MB only- this kind of VM can`t handle standard byte code files .class even compressed files .jar because of its limited size. It needs special pre-processing so it replace .jar .class with classes.dex and apk files. Those kind of files replace every string, every method name and every class name by id and lookup table. This

strategy reduce data loaded in memory and keep more rooms to the actual data in the application.

Android VM A.K.A Sandbox is a tool used in inter-application separation; every application runs in android must running alone on one VM. VM doing inter application division by two ways. First, every app has its different user ID. Second, every app is using its manifest file for to determine specific permissions.

VM Actually opens the gate of reverse engineers to reverse apks to classes.dex and resources. Again, reverse class.dex to classes, which mean inject malware behavior or ads in real and healthy app.

2.1.5 Android Storage Options

Android provides several options for you to save application data. The option you choose depends on your application needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires etc...

Android data storage options are the following: Shared Preferences, Internal Storage, External Storage, SQLite Databases and Network Connection [7]. The Shared Preferences provides a general framework that allows saving and retrieving persistent key-value pairs of primitive data types. Shared Preferences can used to save any primitive data: Booleans, floats, integers, longs, and strings. This data

will persist across user sessions. Internal Storage can used to save files directly on the device's internal storage. By default, files saved to the internal storage are private to your application and other applications cannot access them (nor can the user). When the user uninstalls your application, these files are removed. External Storage are world-readable and can be modified by the user. SQLite Databases used to save structural relation data and retrieve them using SQL standard with integrity constraint and indexing to fast retrieval when there is many data. In addition, android has versioning mechanism to upgrade and downgrade database, which help application to extend their data structure [11]. Network can used (when it is available) to store and retrieve data on your own web-based services.

Android provides a way to expose even your private data to other applications — with a content provider. A content provider is an optional component that exposes read/write access to application data for other applications, subject to whatever restrictions you want to impose.

2.2 Android Malwares

Malicious software is referred to as malware, classified by its nature as either computer virus, Trojan horse, worm, backdoor or rootkit. The most common malware types [12] are:

Virus: Code that that inserts itself into another program and replicates, that is, copies itself and infects other computers. Nowadays often used as a generic term that also includes worms and Trojans horses.

Worm: Self-replicating malware, which copies itself to other nodes in a network without user interaction using vulnerabilities. Worms do not attach themselves to an application like a virus do.

Trojan horse: Malicious program, which masquerades itself as being an application. Unlike viruses and worms, it does not replicate itself.

Rootkit: Software that enables continued privileged access to a computer while actively hiding its malicious activity from administrators by modifying the operating system functionality.

Backdoor: Specialized Trojan horse that masquerades itself as an installed program to enable remote access to a system and bypassing normal authentication. Additionally, backdoors attempts to remain undetected.

Spyware: Software that reveals private information about the user or computer system to eavesdroppers.

Bot: Piece of malware that allows the bot master, i.e. the author to remotely the infected system. Groups of infected systems that are controlled, which are denoted

as botnets, instructed by the bot master to perform various malicious activity such as distributed denial of services, stealing private information and sending spam.

2.3 Related work

Mobile security issues have gained much attention recently. Malware are available on both the official Android market and alternative ones [13]. Research efforts were made on detecting repackaged apps [14] or apps with known malicious behavior [15, 16]. Google also launched its malware filtering engine [17]. Information leakage is another major security threat for mobile devices. Kirin [18] detects apps whose permissions might indicate potential leakage.

In general, information leakage detection reveals the potential out bound propagation of sensitive information, which might be benign in many cases. Instead, component hijacking detection captures the information leakages resulted from an exploitation (i.e. sensitive data theft), in addition to other hijacking types.

Enck et al. introduced Ded [19] to convert Dalvik bytecode back to Java bytecode, and then used existing decompilers to obtain the source code of the apps for analysis.

Android mediates access to protect resources using a permission system. However, it's effectiveness hinges on app developers correctly implementing it. Chin et al. showed that apps might be exploitable when servicing external intents

[20]. They built ComDroid to identify publicly exported components and warn developers about the potential threats. For that, ComDroid checks app metadata and specific API usages.

As a result, warned public components are not necessarily exploitable or harmful (i.e. the openness can be by design or the component is not security critical). On the other hand, Android permission system is subject to several instances of the classic confused deputy attack [21]. As demonstrated by [22 ،23 ، 15], an unprivileged app can access permission-protected resources through privileged apps that do not check permissions. Grace et al. [15] employed an intra-procedural path-sensitive static analysis to discover permission leaks specific to stock apps from multiple device vendors.

Malware detection in general has two track static analysis and dynamic analysis. As Android applications are largely interactive and have a lot of interference between their components, dynamic detecting malware code would face scalability limitations as TaintDroid [24], where authors had to interact manually with each application. This eliminates techniques such as [25, 26, 27, 28] for detecting Android`s malwares applications. Therefore, we concentrate in this research on static analysis.

2.3.1 Static Analysis

Static analysis is an analysis of program application without executing the program. Static analysis of malware android application has three main categorization: Feature Based, Structure Based and Program Dependency Graph (PDG) Based.

2.3.1.1 Feature Based

Feature based approaches analyze a program and extract a set of features. Similarity between program and malware is detecting by comparing the extracted features from the programs. The features choice can vary significantly, from number or size of classes, methods, loops, or variables to included libraries.

Tesfay et Al. [29] Provided Anti-malware cloud that contains reputation for every version of every application using APK's hash code and depends on user Anti-malware reputation. Actually, there approach cannot handle repackaged APKs because simple change like space or comma in the APK content means completely different hash code.

This approach is limited -even with AI still need more investigation [30, 31]- and not realistic because it discards too much information about the structure of the programs.

2.3.1.2 Structure Based

Structure based systems convert programs into a stream of tokens and then compare the streams between two programs. By converting programs into a stream of tokens and ignoring easily, changed constructs such as comments, whitespace, and variable names, structure based systems detect plagiarism more robustly than feature-based systems.

Zhou et Al. [14] They work on DroidMOSS framework, it adopt a specialized hashing technique called fuzzy hashing. Instead of directly processing or comparing the entire (long) instruction sequences, it first condenses each sequence into one much shorter fingerprint. The similarity between two apps calculated based on the shorter fingerprints, not the original sequences.

Even when the does not depend on absolute hash map and replace it with Fuzzy hash map [32, 33] it still face difficulty to detect repackaged apps with small simple refactoring method

Schleimer et Al. [34] they attempt to find plagiarism with modifications using k-grams, by finding common token substrings of length k. If the differences between the programs are relatively infrequent or tend to be greater than k tokens apart then the comparison, will find many k-length token streams in common.

This approach also has a problem because insertion more than k instruction - even when those instruction are naïve and does not modify any behavior or flow- this approach will be failed to detect relation between the produced malware and the original one.

Unfortunately, even when these techniques has result better than feature based, it still vulnerable to addition or deletion of byte code instructions.

2.3.1.3 Program Dependency Graph (PDG) Based

In Apps There are two types of dependencies: data and control. Statement s1 has data dependency on statement s2 if s1 contains variable v, which v value changed in s2. On the other side, statement s1 has control dependency on statement s2 if s2 decide if s1 executed or not.

Crussell et Al. [35] working on detection clones of android apps, they exclude famous libraries as com.facebook.android and com.google.admob using sha1 hash to be sure these libraries untouched. After that, they create PDG for every method and apply losseless and lossy filters for every method pairs in the two apps. Lossless filter removes methods smaller than 10 nodes and lossy filter, which discards method pairs that are unlikely to match due to a difference in the distribution of types of nodes in the two PDGs. After that, they apply VF2

algorithm to compute subgraph isomorphisms. Finally, they calculate similarity of the two application and decide if these apps are clones or not.

This approach is good to determine clones but unfortunately, it has some back doors. First lossless filter can attacke by divide large method to smaller ones. They do not take noisy code in their account. Second it is good for clones but it takes too much time for malware detection and can`t find relation between malware and malicious app

Crussell et Al. in [36] they work on AnDarwin framework its design done on four stages: First, it represents each app as a set of vectors computed over the app's Program Dependence Graphs, split into connected components as multiple data-independent computations. Second, it finds similar code segments by clustering all the vectors of all apps. Third, it eliminates library code based on the frequency of the clusters. Finally, it detects apps that are similar, considering both full and partial app similarity.

CHEX [37] is a tool to detect component hijacking vulnerabilities in Android applications by tracking taints between externally accessible interfaces and sensitive sources or sinks. Although it does not built for the task, CHEX can be used for taint analysis. CHEX does not analyze calls into Android framework itself but instead requires a model of the framework. CHEX's entry-point model requires

an enumeration of all possible “split orderings”. Furthermore, CHEX is limited to at most 1-object-sensitivity.

LeakMiner [38] analyzes Android apps on market site. Thus, it does not introduce runtime overhead to normal execution of target apps. Besides, Leak Miner can detect information leakage before apps are distributed to users, it implements the Android lifecycle but the analysis is not context-sensitive - A context-sensitive analysis is an interprocedural analysis that considers the calling context when analyzing the target of a function call. In particular, using context information one can jump back to the original call site, whereas without that information, the analysis information has to be propagated back to all possible call sites, potentially losing precision-.

AndroidLeaks [39] also state the ability to handle the Android Lifecycle including callback methods. It is based on WALA’s context-sensitive System Dependence Graph with a context-insensitive overlay for heap tracking, but it taints the whole object if tainted data is stored in one of its fields, i.e., is neither field nor object sensitive. This precludes the precise analysis of many practical scenarios.

SCanDroid [40] is a tool for reasoning about data flows in Android applications. Its main focus is the inter-component (e.g. between two activities in

the same app) and inter-app data flow. This poses the challenge of connecting intent senders to their respective receivers in other applications. SCanDroid prunes all call edges to Android OS methods and conservatively assumes the base object, the parameters, and the return value to inherit taints from arguments.

EPICC [41] proposes a string analysis for inferring inter component communication specifications. These include inter component communication entry and exit points, information about the action, data and category components of intents used for inter component communication, as well as Intent key/value types.

2.4 Summary

There is many research efforts on Android malware detection, repackaging app detection, cloning apps detection and leakage information detection. They cover inter-process communication, permission up-used, information leakages and harmful operation but they do not discuss the idea of leakage information on multi app version. On our research, we will discuss this idea and create a tool to detect these leakages.

العنوان:	Detection Of Redestributed Malware Behavior In Android App Versions
المؤلف الرئيسي:	Al Salehi, Alaa
مؤلفين آخرين:	Abu Samra, Aiman Ahmed(Advisor)
التاريخ الميلادي:	2015
موقع:	غزة
الصفحات:	1 - 89
رقم MD:	768547
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	الجامعة الإسلامية (غزة)
الكلية:	كلية الهندسة
الدولة:	فلسطين
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسوب، البرمجيات الخبيثة، برنامج أندرويد
رابط:	https://search.mandumah.com/Record/768547

3- Research Tools

Malwares for Android application are considered as one of the most growing problems, so there must be a new techniques and tools to detect these malwares. In fact there are many antiviruses' tools in today market to detect malware using either static or dynamic analysis, In this chapter a scientific view will be presented for algorithms and techniques used for this research.

3.1 Reverse Engineering

Reverse Engineering is a process of analyzing program code or software in order to test it from any vulnerability or any errors. Reverse engineering is the ability to generate the source code from an executable code. This technique is used to examine the functioning of a program or to evade security bugs, etc. Reverse engineering can therefore be stated as a method or process of modifying a program in order to make it behave in a manner that the reverse engineer desires.

Joany Boutet has quoted Shwartz, saying,

"Whether it's rebuilding a car engine or diagramming a sentence, people can learn about many things simply by taking them apart and putting them back together again. That, in a nutshell, is the concept behind reverse engineering -breaking something down in order to understand it, build a copy or improve it " [42]

From the beginning of 2009, research scientists began proposes tools for reverse the DalvikBytecode. One of them is undX tool which could generate a JAR

file from an Android APK file, then convert to JAVA using tools such as JAD and and JD-GUI. The undX tool worked well with basic applications; but it posed many problems when dealing with complex Dalvik Bytecode. The Dex2Jar tool originated then. Dex2Jar does similar job to undX; but this tool also has some issues while dealing with complex Dalvik Bytecode.

The application, in its pre-compiled binary format, is distributed and hence it is not possible to directly debug the source code but there are disassemblers that convert or reverse the Dalvik Bytecode into readable format. The binaries for Dalvik Virtual Machines are in the .dex format. Backsmali [43] is a disassembler that is used for .dex files in Dalvik VM. Backsmali convert .dex file to intermediate language with full support of .dex and without lose anything.

3.2 Static Analysis

Static analyses inspect code to derive information about the application's behavior at runtime. Every application has variables (inputs from a user, files, internet etc.) an analysis has to abstract from concrete program runs. Static analyses aims to cover all possibilities by making assumptions. The properties derived from these assumptions can be weaker than the program's properties actually are, but they are guaranteed to be applicable for every program run. In this way, static analysis detects an application behavior, which might not actually

happen during runtime, but it does not miss a behavior, which can happen during runtime (i.e. privacy invasion).

In general, there are two different approaches to static analysis: type systems and data-flow based approaches. Type systems assign properties to components of the application and checks whether they are going to hold during run time. Data-flow based is a technique for gathering information about the possible set of values calculated at various points in an application.

Modern sophisticated tools convert the input (either bytecode or source code) to intermediate representations on which they can efficiently operate. To model the program flow they create control-flow graphs (CGF) and call graphs. CGF represent intra-procedural sequences of statements, call graphs contain edges between a call site and the call target.

Usually it is not possible to determine these targets unambiguously: The method invoked by the call site can refer to the implementation of the class specified in the call site or any other subclass. For example, a class A defines the method m() and has a subclass B. The call site x.m() can either refer to the implementation of A or B, depending on the initialization of x, which might not be statically resolvable.

3.2.1 Call graph (CF)

A call graph is a directed graph that represents calling relationships between functions in a computer program. Specifically, each node represents a function and each edge (f, g) indicates that function f calls function g. Thus, a cycle in the graph indicates recursive function calls.

Call graphs are a basic program analysis result that can be used for human understanding of programs, or as a basis for further analyses, such as an analysis that tracks the flow of values between functions. One simple application of call graphs is finding functions that are never called.

A static CG is a call graph intended to represent every possible run of the program. The exact static CG is an undecidable problem, so static call graph algorithms are generally over-approximations. That is, every call relationship that occurs is represented in the graph, and possibly some call relationships that would never occur in actual runs of the program. CG can be resource consumers in construction process, visiting call nodes and memory storage or example, constructing the CG of a Java “Hello, World!" program using Spark [44] can take up to 30 seconds, and produces a CG with 5,313 reachable methods and more than 23,000 edges. Because of that, CG can be defined to represent varying degrees of precision. A more precise CG more precisely approximates the behavior of the real program, at the cost of taking longer to compute and more memory to store. The

most precise CG is fully context-sensitive, which means that for each function, the graph contains a separate node for each call stack that function can be activated with. A fully context-sensitive CG is called calling context tree. A calling context tree can be computed dynamically easily, although it may take up a large amount of memory. Calling context trees are usually not computed statically, because it would take too long for a large program. The least precise call graph is context-insensitive, which means that there is only one node for each function. This is a tradeoff problem will be shown in the following example

```
public class Example1 {
    public static void main(String[] args) {
        String s1=newLine("hello");
        String s2=newLine("world");
        System.out.println(s1.concat(s2));
    }

    public static String newLine(String input)
{
        if(input.equals("hello"))
            return tab(input.concat("\n"));
        else
            return input.concat("\n");
    }

    public static String tab(String input) {
        return input.concat("\t");
```

```
    }
}
```

Code1 is simple java application contains three method to clarify context sensitivity levels no context sensitivity every method has only one node for method in simple words there is no different between call happened on tab call in newline and call happened on newline. On the other hand, context sensitive CG has node for every method call happened this gives more information about the context this method called on it.

3.3 WALA

Watson Libraries for Analysis (WALA) is a framework provides static and dynamic analysis capabilities for Java bytecode and related languages and for JavaScript. WALA is licensed under the Eclipse Public License. The initial WALA infrastructure was independently developed as part of the DOMO research project at the IBM T.J. Watson Research Center. In 2006, IBM donated the software to the community.

Core WALA Features

WALA features include:

1- Java type system and class hierarchy analysis

2- Source language framework supporting Java and JavaScript

3- Interprocedural dataflow analysis (RHS solver)

4- Context-sensitive tabulation-based slicer

5- Pointer analysis and call graph construction

   a. Several algorithms provided (RTA, variants of Andersen's analysis)

   b. Highly customizable (e.g., context sensitivity policy)

      i. ZeroCFA context insensitive

      ii. ZeroOneCFA context sensitive

   c. Tuned for performance (time and space)

6- Static single assignment form SSA-based register-transfer language IR

   a. SSA exsit on wala for Java and Java Script

   b. Anyone can extend it and IR for other languages

7- General framework for iterative dataflow

8- General analysis utilities and data structures

9- A bytecode instrumentation library (Shrike) and a dynamic load-time instrumentation library for Java (Dila).

10- Robustness, Efficiency and Extensibility.

3.4 Scandroid

SCanDroid [40] is a tool for reasoning about data flows in Android applications. Its focus is the inter-component (e.g. between two activities in the same app) and inter-app data flow. This poses the challenge of connecting intent

senders to their respective receivers in other applications. SCanDroid prunes all call edges to Android OS methods and conservatively assumes the base object, the parameters, and the return value to inherit taints from arguments.

Scandroid one of the first tools created to static analysis for android apps they tried to create automatic security certification for android application. SCANDROID's analysis is modular to allow incremental checking of applications as they are installed on an Android device. It extracts security specifications from manifests that accompany such applications, and checks whether data flows through those applications are consistent with those specifications.

They converted android byte code of delvaik VM to SSA-instruction compatible with intermediate representation (IR) form of WALA and make WALA framework able to use for static analysis for android applications. They make their source available on github.

We use their conversion of android byte code to IR, which is perfect, and used by almost all static analysis tools using WALA framework for android static analysis.

3.5 Summary

This chapter describes in theoretical view of the most important used tools in this thesis; First a scientifically description about static analysis and call graph

concepts has been discussed then a complete description about WALA framework with its usage and features has been provided and finally information about Scandroid and its implementation for android delvik byte code to WALA IR.

Those tools are used with reverse engineering as tool to insure results of these tools are precise and correct.

4- Methodology Evaluation and Analysis

In this chapter a methodology, experiments and thesis proposed algorithm will be discussed. In Section 5.1 discuss our proposed algorithm named Distributed Malware Detection Algorithm we create to solve leakage information over versions of application, Section 5.2 discuss implementation and used entry points, sources, sinks, transient sinks and transient sources , Section 5.3 is the experiment and evaluation, . Finally, section 5.4 is Summary for chapter.

4.1 DMDA Algorithm

In this thesis, we propose a new method to detect malwares distributed over application versions. Versioning will help malwares, which are leak information - Appendix A have example- and malwares can be divided to steps. The algorithm proposed will help to detect these malwares using call graph and pointer analysis.

4.1.1 Definitions

Definition 1 (Entry Point)

العنوان:	Detection Of Redestributed Malware Behavior In Android App Versions
المؤلف الرئيسي:	Al Salehi, Alaa
مؤلفين آخرين:	Abu Samra, Aiman Ahmed(Advisor)
التاريخ الميلادي:	2015
موقع:	غزة
الصفحات:	1 - 89
رقم MD:	768547
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	الجامعة الإسلامية (غزة)
الكلية:	كلية الهندسة
الدولة:	فلسطين
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسوب، البرمجيات الخبيثة، برنامج أندرويد
رابط:	https://search.mandumah.com/Record/768547

concepts has been discussed then a complete description about WALA framework with its usage and features has been provided and finally information about Scandroid and its implementation for android delvik byte code to WALA IR.

Those tools are used with reverse engineering as tool to insure results of these tools are precise and correct.

4- Methodology Evaluation and Analysis

In this chapter a methodology, experiments and thesis proposed algorithm will be discussed. In Section 5.1 discuss our proposed algorithm named Distributed Malware Detection Algorithm we create to solve leakage information over versions of application, Section 5.2 discuss implementation and used entry points, sources, sinks, transient sinks and transient sources , Section 5.3 is the experiment and evaluation, . Finally, section 5.4 is Summary for chapter.

4.1 DMDA Algorithm

In this thesis, we propose a new method to detect malwares distributed over application versions. Versioning will help malwares, which are leak information - Appendix A have example- and malwares can be divided to steps. The algorithm proposed will help to detect these malwares using call graph and pointer analysis.

4.1.1 Definitions

Definition 1 (Entry Point)

An Entry point is the point where operating system enters a program. In many programming languages, the main function is where a program starts its execution. Android is operating system with multiple entry point. Activity onCreate method is entry point.

Definition 2 (Source)

Source is calls into resource method returning non-constant value into the application code. This value is valuable to user privacy or user life. Example getDeviceId() resource method is an Android source. It returns a value (the IMEI) into the application code.

Definition 3 (Sink).

Sink is calls into resource method accepting at least one non-constant data value from the application code as parameter, if and only if those parameters go out the application. The sendTextMessage() resource method is an Android sink as the message text are possibly non-constant and goes to phone number.

Definition 4 (Transient Source)

Transient Source is calls into resource method returning non-constant value stored into local storage to the application code. This value is valuable to user privacy or user life. Example retrieve data from local database.

Definition 5 (Transient Sink)

Transient Sink is calls into resource method accepting one non-constant data value from the application code as parameter if and only if those parameters goes to local storage resource. Example saving contacts information on local database.

Definition 6 (Leak)

Leak is a call graph path where start in Source resource and end to Sink resource. Example Application send contacts data to internet website.

Definition 7 (Transient Leak)

Transient leak is a call graph path where start in Source resource and end to transient Sink resource. Example Application save contacts data into local storage media.

4.1.2 Attack model (Distributed Malware Attack Model)

Android is an open environment for development but this make it a field for malwares as demonstrated by [22 ،23 ،15]. Those researchers talked about misuse permissions, Exploiting over permissions by malicious applications and data leaks. Many researcher efforts talking about it as mentioned on chapter 2. Their efforts is focused on one version of android application. But android OS and many of android markets including Google Play market support versioning on android application [45] through android-mainfest.xml attributes android:versionCode and

android:versionName so it is simple to malware producer to distribute its malware on multi-version of android application. Version one got the data from android OS ex. contacts and SMSs and stored it on its own data, and on version two remove the code was responsible to store these data and add other code which is misuse these information like leak these data to internet.

4.1.3 DMDA Algorithm

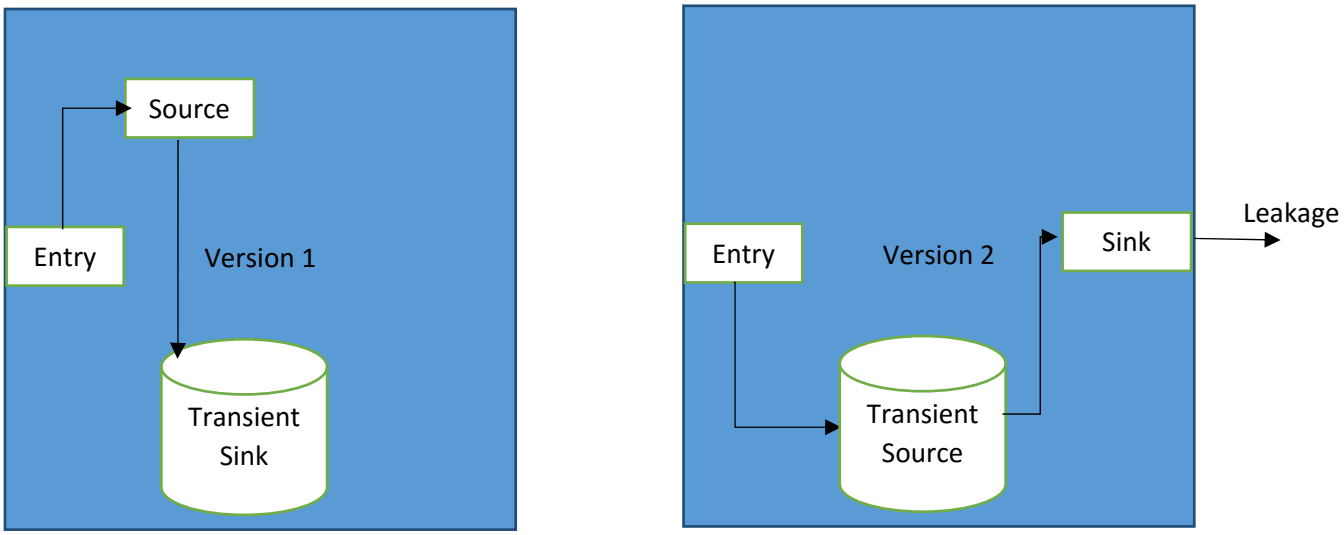

Figure 3: Attack Model

In the section we describe DMDA algorithm created to detect, malware apps distribute their malware behavior on their versions So these application split their Source and sink methods into two versions or more using transient sink and transient source.

Figure 3: Attack Model shows attack model, DMDA algorithm design to detect. This model distributed its source and sink into two different versions of the app.

DMDA algorithm steps are

1- Determine entry points of app version
2- Create call graph cg based on S where S is group of E and E is an entry point. Cg will contains N where N is a group of nodes and D where D is a group of edges
3- Visit call graph and determine PI –pure sink- , PO –pure source- , transient TI –transient sink- and TO –transient Source- nodes these nodes.
4- Using [47] to solve dependencies and reduce reachability.
5- if v == 1 then where v is the version of app
    a. call findLeak
    b. call findTransientSink
6- if transient think exist then
    a. call savetransientSink
7- if v>1
    a. call findTransientLeak

b. call findLeak

c. call findTransientSink

d. saveTransientSink

findLeak procedure

1- if there is path between source and sink then leak is exist and this is a malware app

saveTransientSink procedure

1- after finding transient sink check for possible paths used for this sink if there is path to source this path will saved
2- save the key used to this parameter –example table name for inset statement -

findTransientLeak procedure

1- if transient sink saved before have the same key of transient sink then this transient leak do
    a. replace transient sink with transient source paths stored before
2- else
    a. ignore this transient source

Figure 4: DDMA Algorithm shows the steps of DMDA algorithm in a simple way represent the model present on attack model in Figure 3: Attack Model and shows main idea of distribution and where main steps of algorithm happened.

Figure 5: DDMA Algorithm`s Flow shows steps of DMDA algorithm in simple flowchart diagram. The diagram focused on main steps of algorithm like transient sink and transient source.

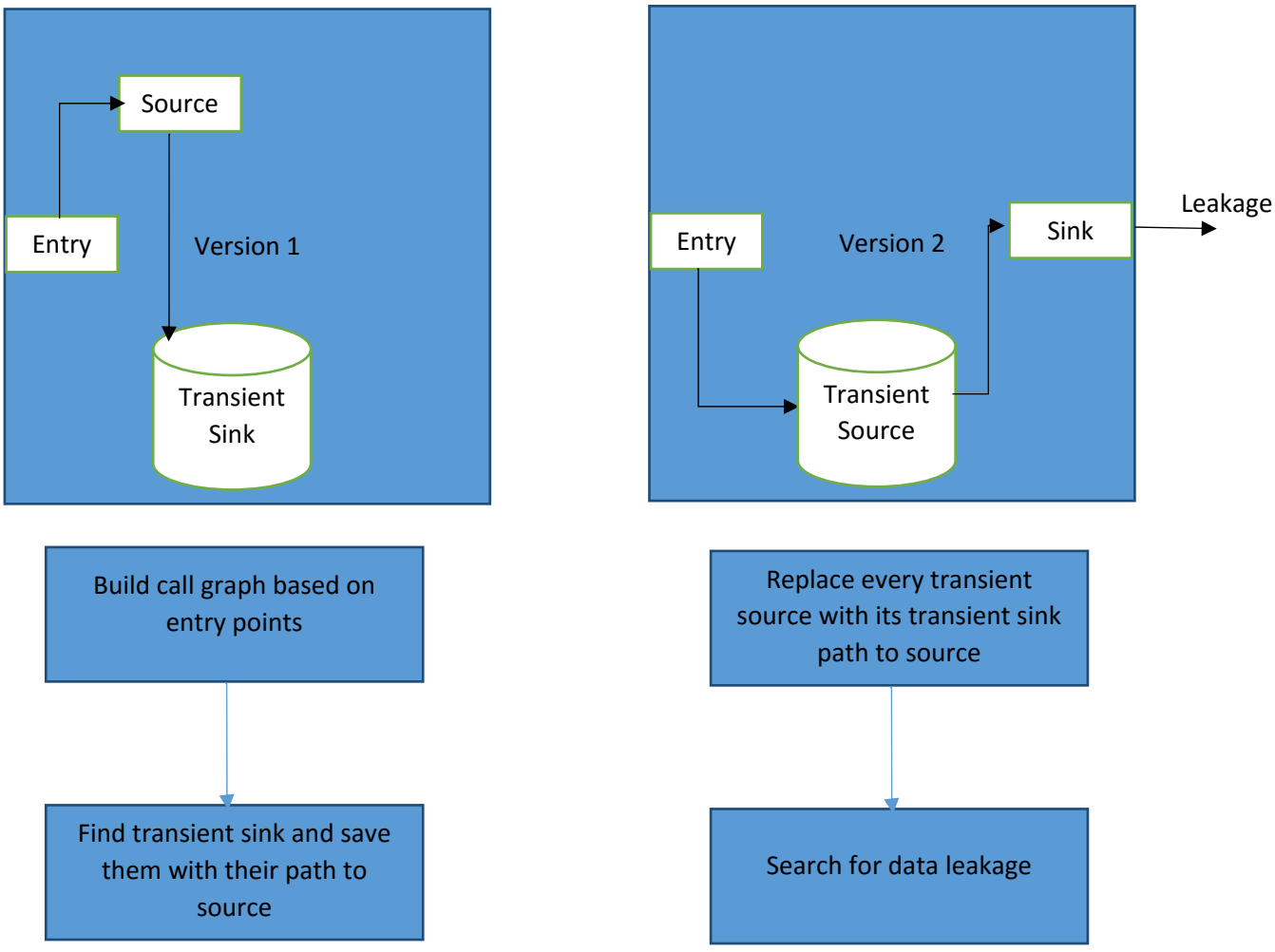

Figure 4: DDMA Algorithm`s Model

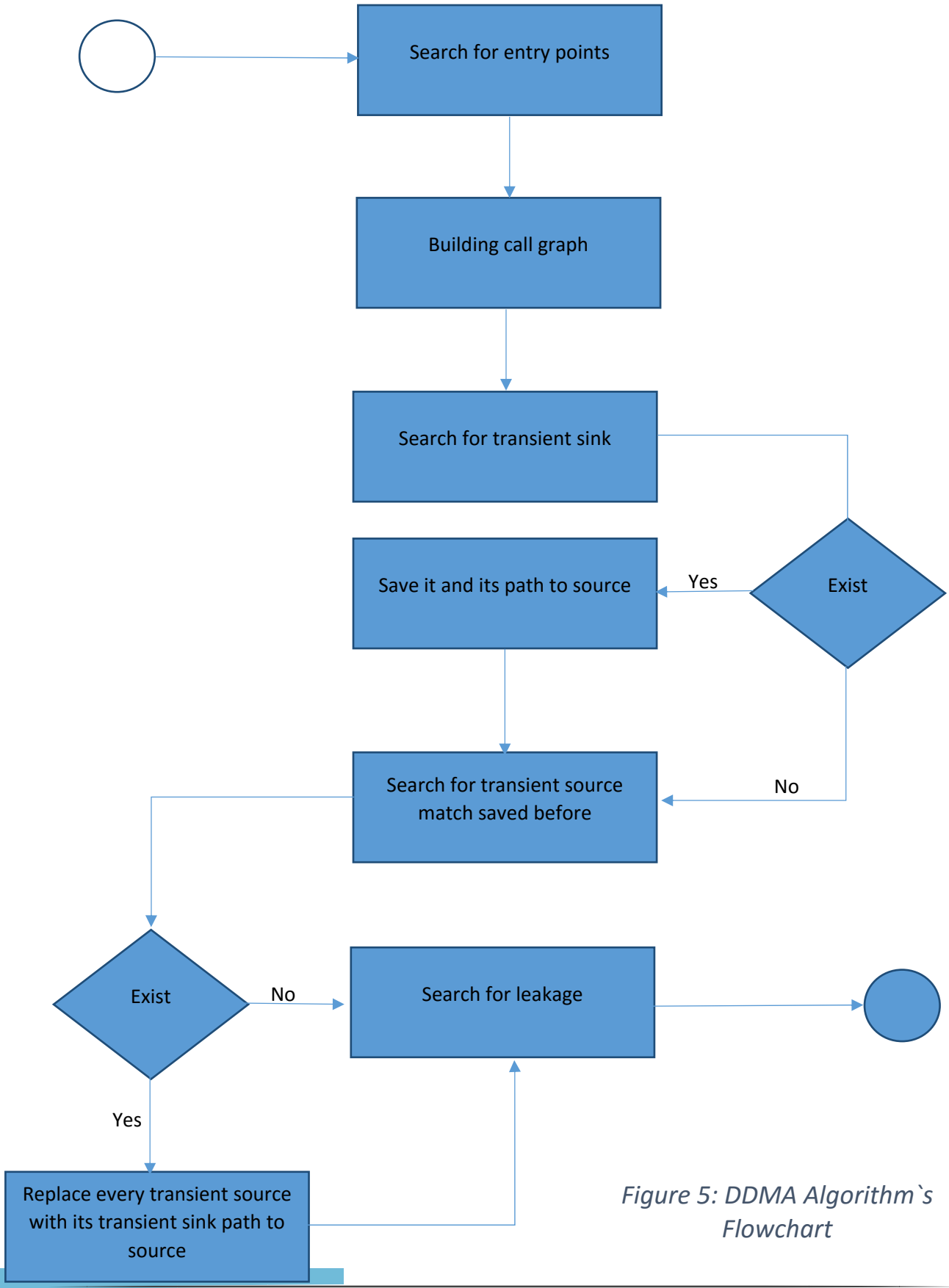

Figure 5: DDMA Algorithm`s Flowchart

4.2 Implementation

This chapter explain DMDA algorithm implementation, entry points for android application, sources, sinks, transient sinks and transient sources and how WALA frameworks used to implement DMDA algorithm. The real code attached on Appendix B to more details.

4.2.1 Android Entry points

Android is not like java android have multi-entry point those entry points are the real start of building android app call graph. Entry points of android are:

1- Android.app.Activity: onCreate, onStart, onRestart, onResume, onPause, onStop, onDestroy, onActivityResult, onRestoreInstanceState, onSaveInstanceState.

2- Android.app.Service: onCreate, onStart, onStartCommand, onDestroy, onBind.

3- ContentProvider: onCreate, query, insert, update, delete.

4- BroadcastReceiver: onReceive.

Those are the entry points Android OS can start android application from them and those are the seed of call graph.

4.2.2 Source and Sink

After determine entry points, must determine source and sink of our analysis. Source and sink concept is a famous concept in privacy and data leak analysis where data is a precious. Source is a method return a valuable data like phone number, user SMSs, contacts, browser history etc…. Sink is a method leak these data or misuse like send it over internet, SMS, Bluetooth etc…

For this research, we choose these Sources

1- Android.content.ContentResolver. query: this method used to query any content provider on android app. It is one of the most famous sources on android. Any developer with the right permission can query SMS, Contacts, browsing history and other app data.
2- Android.location.LocationManager.[all methods]: this class is responsible to location stuff contains GPS providers and location and last known plcae.
3- Android.telephony.TelephonyManager.[getNeighboringCellInfo| getCellLocation]: these methods return data about GSM cells these information can leak user location.

In addition, these Sinks:

1- Android.app.Activity.setResult: this method used to respond on call of startActivityForResult and it can leak data on it is parameter to other android application.

2- Android.app.Activity.[starActivity| starActivityForResult| startActivityIfNeeded| startNextMatchingActivity| startActivityFromChild]: these methods can leak data to other application on the intent send to start their activities.
3- Android.content.ContentResolver.[query|insert|update|delete]: these methods help developers to access Content Provider query, insert, update and delete
4- Android.telephony.gsm.SmsManager.[sendTextMessage|sendDataMessage| sendMultipartTextMessage]: those methods can leak data through GSM messages.
5- Android.net.AndroidHttpClient.execute: this method can leak data through it is parameter to internet.
6- java.net.HttpURLConnection.[getOutputStream| setRequestProperty]:these methods can leak data through http request on header or body.
7- java.net.CookieManager. setCookie: this method can leak data through http header called cookies.

Those are not all the sources and sinks on android those are the ones used on our research to prove the idea there is other research doing hard work on this point [24, 19, 40].

4.2.3 Transient Sources and Sinks

Transient sources and sinks are those methods used to store application data into local storage. Those cannot consider as a pure sources and pure sinks because they almost used to store clear data so considering them as a source or sink probably result a false positive malware detection. But ignoring them lead to miss malwares divided into application versions.

For this research, we choose the following method as transient Sink:

1- android.content.SharedPreferences.Editor[putBoolean| putFloat| putInt| putLong| putString| putStringSet]: shared preference used to persistent primitive data or Strings and reuse them after a while. These methods can transiently leak data through their second parameter.
2- android.database.sqlite.SQLiteDatabase[insert| insertOrThrow| insertWithOnConflict| replace| replaceOrThrow| update| updateWithOnConflict]:SQLLite is a simple relational database used to store complex data types and reuse them with fast query. These methods can transiently leak data through their second parameter through their parameter ContentValues.

For this research, we choose the following method as transient Source:

1- android.content.SharedPreferences [getBoolean| getFloat| getInt| getLong| getString| getStringSet]: these methods used to retrieve data stored on put methods. These methods can transiently been a source of data through their return values.

2- android.database.sqlite.SQLiteDatabase[query| queryWithFactory| rawQuery| rawQueryWithFactory]: those methods used to retrieve data stored using update, insert and replace methods. These methods can transiently been a source of data through their return values.

All these lists –sources, sinks, transient sinks, transient sources and entry points- included on eAndroidSPec.java, which is, extend ISpec class one of scandroid specifications.

4.2.4 Exclusion list

As described on this research call graph and static analysis is greedy for memory even simple ones can take too much memory. Because of that WALA have exclusion list, which used to exclude unimportant classes from call graph and data flow analysis. We exclude famous used libraries and basic java packages. This technique help WALA to reduce memory and increase productivity. The full list of excluded packages in Appendix B.

4.2.5 Implementation

Using WALA Framework help in implementation a lot of algorithm implementing and available to extend and reuse. We use WALA call graph which depends on graph reachability concept and Pointer analysis implementation using kidall`s Framework [49] to follow keys of transient source and transient sinks.

Kidall`s Framework based on simple constant propagation this idea was created firstly by Kidall in [49] to Discover values that are constant on all possible executions, and propagate values.

Simply this algorithm steps are start on entry point, process this entry point and produce constant propagation, send these information to all first successor of this entry point, repeat this in next successors, merge these information –intersection them- if the data on variable is different on two branches this variable are not included on data return by the algorithm.

This algorithm used into extract keys used to store data in transient sink and transient source.

Based on these algorithms we build two filters: Leakage Filter, which responsible to detect leakage malware behavior and transient filter, which responsible to detect transient leakage and replace transient nodes with the original code.

Transient leakage filter build call graph with context sensitivity for string objects after building call graph this filter search for transient source and check previous list of transient sink if the key of transient source equal one of the these keys it replace the statement with call graph saved for that key finally the filter search for the transient sink and store them with their pruned call graph.

Leakage filter take the call graph built in transient leakage filter, start searching of paths connect source, and sink if there is a path or more then this filter recognize this app as a malware.

4.3 Experiment: Malware detection

In this section, two experiments made to show and explain the attack and experiment effectiveness of DDMA Algorithm. First experiment focus on the attack model and how distribution of a famous malware in two versions make most of anti-malware blind. The second experiment check effectiveness of DMDA algorithm to find malware behavior distributed over android app versions and check these apps.

4.3.1 Attack Model Expirment

Therefore, we think any malware distributed on app versions make most of anti-malware blind even for simple, old and famous malwares like DroidKungFu [46]. DroidKungFu is a Trojan, which although seemingly inoffensive, can actually carry out attacks and intrusions: screen logging, stealing personal data, etc. We use

DroidKungFu as example to explain the attack model. Appendix A contains the DroidKungFu on two versions we test the two versions on VirusTotal –which is a free online service subsidiary of Google that analyzes files and URLs enabling the identification of viruses, worms, trojans and other kinds of malicious content detected by antivirus engines and website scanners – and no anti-malware of 57 scanning the APKs catch the malware.

4.3.2 Effectiveness of DDMA

For this experiments chosen group of apps include the app in Appendix A with two versions of every app these apps taken from [50].

This group contains 100 apps all of them related to contact APIs for every app two chosen versions in this sample next table show the results

No. of apps	No. of versions for app	Transient sources	Transient sinks	sources	leakages
100	2	156	209	200	2

We find over 200 transient sinks and over 150 transient sources these are not a leakage but these may turn on future to leakage. We find also two leakages.

We check the versions where leakage happened by using reverse engineering tools discussed before what we found is interesting one is contains the problem the other one was false positive because of SQL complications.

Select mydata from ComplexMyDataAndContactsData;

This was the reason of false positive we have. We will discuss this issue in future work.

Also we got attention that some apps stored unimportant data like contact id or contact created time these will be considered on our application as a leakage but these values does not have any valued so developers may send it without mean to leakage data.

العنوان:	Detection Of Redestributed Malware Behavior In Android App Versions
المؤلف الرئيسي:	Al Salehi, Alaa
مؤلفين آخرين:	Abu Samra, Aiman Ahmed(Advisor)
التاريخ الميلادي:	2015
موقع:	غزة
الصفحات:	1 - 89
رقم MD:	768547
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	الجامعة الإسلامية (غزة)
الكلية:	كلية الهندسة
الدولة:	فلسطين
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسوب، البرمجيات الخبيثة، برنامج أندرويد
رابط:	https://search.mandumah.com/Record/768547

الملخص

مع زيادة أهمية البيانات الموجودة على الأجهزة الذكية زادت البرمجيات الخبيثة التي تحاول الحصول على هذه المعلومات لاستغلالها إما عن طريق تحليل البيانات وتعقب أصحاب الأجهزة لأغراض إعلانية أو أغراض تسويقية أو عن طريق ابتزاز المستخدمين. وزيادة هذه البرمجيات أدت إلى زيادة في أهمية الأبحاث التي تعمل على تحليل البرمجيات الخبيثة واكتشاف هذا النوع من السلوك.

هذه الأطروحة تدرس تحوير السلوك الخبيث الذي يتم فيه توزيع مصدر البيانات ونقطة فقد البيانات على الإصدارات المختلفة من البرنامج باستخدام وسائل التخزين المحلية في البرامج لتخزين جزء أو كل من هذه البيانات الحيوية.

قامت هذه الأطروحة ببناء خوارزمية سميت Distributed Malware Detection Algorithm واختصاراً DMDA هذه الخوارزمية تقوم بتحليل البيانات وتحديد مصادر البيانات ونقاط الفقد الانتقالية التي يتم استخدامها للتعتيم على عملية كشف مناطق فقد البيانات على مستوى أكثر من نسخة من البرنامج.

تمت تجربة هذه الخوارزمية على عينة من تطبيقات الأندرويد المنشورة على سوق Google Play تحتوي على 100 تطبيق كل تطبيق تم أخذ نسختين منه وتم كشف عما يزيد 150 مصدر انتقالي للبيانات و200 نقطة فقد للبيانات انتقالي ونقطة فقد وحيدة وتم تجريب البرامج نفسها بإصداراتها على 56 مضاداً للأكواد الخبيثة ولم تجد أياً منها أي مشكلة في أي من إصداراتها.

Abstract

The importance of data stored on smart devices can make malware apps that are trying to get this information to be exploited in either the data analysis and tracking devices for the purposes of the owners of advertising or marketing purposes or for blackmailing purposes of users. Increasing malwares has led to an increase in the importance of research work on malware analysis and the discovery of this kind of behavior.

This thesis is considered altered attack method, which distribute of the data source and the point of loss of data on different versions of the app using local storage to storing part or all of vital data to leak in future.

This thesis will introduce Distributed Malware Detection Algorithm (DMDA), which is an algorithm to detect distributed malware on app versions and propose new way to analyze application against redistributed malware.

DMDA created to analyze the data and identify transitional losses points that are used to gloss over the algorithm sources.

We test this algorithm on a sample of Android applications published on the Google Play market containing 100 application; every application has two version of it. The algorithm was revealed 150 transient data source, 200 transient loss of data point and 2 leakage of data. This dataset was checked by 56 anti-malware and none of them find any malicious code.

العنوان:	Detection Of Redestributed Malware Behavior In Android App Versions
المؤلف الرئيسي:	Al Salehi, Alaa
مؤلفين آخرين:	Abu Samra, Aiman Ahmed(Advisor)
التاريخ الميلادي:	2015
موقع:	غزة
الصفحات:	1 - 89
رقم MD:	768547
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	الجامعة الإسلامية (غزة)
الكلية:	كلية الهندسة
الدولة:	فلسطين
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسوب، البرمجيات الخبيثة، برنامج أندرويد
رابط:	https://search.mandumah.com/Record/768547

Contents

العنوان:	Detection Of Redestributed Malware Behavior In Android App Versions
المؤلف الرئيسي:	Al Salehi, Alaa
مؤلفين آخرين:	Abu Samra, Aiman Ahmed(Advisor)
التاريخ الميلادي:	2015
موقع:	غزة
الصفحات:	1 - 89
رقم MD:	768547
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	الجامعة الإسلامية (غزة)
الكلية:	كلية الهندسة
الدولة:	فلسطين
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسوب، البرمجيات الخبيثة، برنامج أندرويد
رابط:	https://search.mandumah.com/Record/768547

اكتشاف السلوك الخبيث الموزع على عدة إصدارات من برنامج أندرويد

DETECTION OF REDESTRIBUTED MALWARE BEHAVIOR IN ANDROID APP VERSIONS

By

Alaa Al Salehi

120090707

A Master of Science Thesis Proposal

Supervisor:

Dr. Aiman Abu Samra

Computer Engineering Department

Islamic University

Gaza

Palestine

Feb, 2015

الملخص

مع زيادة أهمية البيانات الموجودة على الأجهزة الذكية زادت البرمجيات الخبيثة التي تحاول الحصول على هذه المعلومات لاستغلالها إما عن طريق تحليل البيانات وتعقب أصحاب الأجهزة لأغراض إعلانية أو أغراض تسويقية أو عن طريق ابتزاز المستخدمين. وزيادة هذه البرمجيات أدت إلى زيادة في أهمية الأبحاث التي تعمل على تحليل البرمجيات الخبيثة واكتشاف هذا النوع من السلوك.

هذه الأطروحة تدرس تحوير السلوك الخبيث الذي يتم فيه توزيع مصدر البيانات ونقطة فقد البيانات على الإصدارات المختلفة من البرنامج باستخدام وسائل التخزين المحلية في البرامج لتخزين جزء أو كل من هذه البيانات الحيوية.

قامت هذه الأطروحة ببناء خوارزمية سميت Distributed Malware Detection Algorithm واختصاراً DMDA هذه الخوارزمية تقوم بتحليل البيانات وتحديد مصادر البيانات ونقاط الفقد الانتقالية التي يتم استخدامها للتعتيم على عملية كشف مناطق فقد البيانات على مستوى أكثر من نسخة من البرنامج.

تمت تجربة هذه الخوارزمية على عينة من تطبيقات الأندرويد المنشورة على سوق Google Play تحتوي على 100 تطبيق كل تطبيق تم أخذ نسختين منه وتم كشف عما يزيد 150 مصدر انتقالي للبيانات و200 نقطة فقد للبيانات انتقالي ونقطة فقد وحيدة وتم تجريب البرامج نفسها بإصداراتها على 56 مضاداً للأكواد الخبيثة ولم تجد أياً منها أي مشكلة في أي من إصداراتها.

Abstract

The importance of data stored on smart devices can make malware apps that are trying to get this information to be exploited in either the data analysis and tracking devices for the purposes of the owners of advertising or marketing purposes or for blackmailing purposes of users. Increasing malwares has led to an increase in the importance of research work on malware analysis and the discovery of this kind of behavior.

This thesis is considered altered attack method, which distribute of the data source and the point of loss of data on different versions of the app using local storage to storing part or all of vital data to leak in future.

This thesis will introduce Distributed Malware Detection Algorithm (DMDA), which is an algorithm to detect distributed malware on app versions and propose new way to analyze application against redistributed malware.

DMDA created to analyze the data and identify transitional losses points that are used to gloss over the algorithm sources.

We test this algorithm on a sample of Android applications published on the Google Play market containing 100 application; every application has two version of it. The algorithm was revealed 150 transient data source, 200 transient loss of data point and 2 leakage of data. This dataset was checked by 56 anti-malware and none of them find any malicious code.

Dedication

To my parents, my family, my wife and to my baby Mohamed

Acknowledgement

I would like to acknowledge my thesis supervisors Dr. Aiman Abu Samra for his guidance and valuable help. I also want to acknowledge my college Mahmoud Al-kurdi who is help me with valuable resources.

Contents

Figures

1- Introduction

1.1 Topic Area

Smart phones are becoming more integrated and important part of people's daily lives due to their highly powerful computational capabilities, such as email applications, online banking and online shopping…etc.

Malware, short for malicious software, is one of the major security threats in information systems. Malware includes viruses, worms, Trojan horses, spyware, dishonest adware, most root kits, and other malicious and unwanted software [1].

Android is an OS for smart phone owned by Google Inc, Google wants Android to become dominant in smart phone field, so they create their market to be an open market for developers with easy conditions for publishing new apps. In addition, Google opened Android for company solutions –companies can deploy their own modification on Android OS, Also Google allows Android`s users to install apps from other markets –there is a lot of android markets like Amazon store, SildeMe, Aptoide,…etc - and even form a website –unknown source-.This makes android a great environment for developers, marketers, users and companies.

This tremendous increase unfortunately, also makes android target for Malware applications and application`s thieves. Malware applications become the main threat field because of large custom and private data can be collected form user smart phones like Identifiers Disclosure - individually phone number, International

Mobile Equipment Identity number (lMEI)-, SMS, call log , contacts, browser history, location and emails. In addition, Malware can misuse SMS for Premium messages and root exploits. [2, 3, 4, 5]

In addition to malware android is a hot business field for developers also repackage app can threat their businesses. There are several ways developers may lose potential revenue: a paid application may be “cracked” and released for free, a free application may be copied and re-released on other markets with changes to the ad libraries or even in the same market with changes on interface and services. That will cause ad revenue or paid price goes to the plagiarist.

1.2 Research Question

The popularity and adoption of Smart phones has greatly stimulated the spread of mobile malware, especially on the popular platforms such as Android. In light of their rapid growth, there is a pressing need to develop effective solutions. General countermeasures to Android malwares are currently limited to signature-based antivirus scanners, which efficiently detect known malwares, but they have serious shortcomings with repackaged, refectories and redistributed. These maybe on threads, on versions, on components or maybe on different applications.

So the question is how to detect these behaviors on apps?

1.3 Significance

Tremendous increase of android markets make it easy for anyone to publish apps and update these apps. There is also a rising danger associated with Malware applications at mobile devices, so the problem of detecting Malwares is an interesting topic. In fact, 86% of detected malwares are old malware repackaged in new apps [6]. However, the fact all antimalware and antivirus focus on the current app version and they do not count malwares distributed on different versions of the same application.

In this research, we introduce a way to detect distributed malware on app version application and propose new way to analyze application against redistributed malware.

1.4 Thesis Structure

This thesis is organized as follows:

Chapter 1; Introduction: In this chapter thesis provides an introduction about thesis problem, questions and significance, this chapter describes why we choose this title for thesis and the idea of proposed solution.

Chapter 2; Background and Related Work: This chapter provides a background about Android system, application and programming. It also talks about malwares in general and malware in Android applications, at the end of this chapter there is a group of related work in the same topic of this thesis.

Chapter 3; Research Approach and Tools: This chapter describes in theoretical view the most important used tools in this thesis; it provides readers with description about used applications.

Chapter 4; Attack model: This chapter describes the model assumed on attack and the idea of distribution malware behavior.

Chapter 5; Methodology Evaluation and Analysis. Here readers can show the used methodology for thesis, and how we prove the feasibility of our idea, details of DMDA algorithm, this chapter also provides details about experiments and it results, in addition, it provides more details about algorithm.

Chapter 6; Conclusion and Future Work: A complete conclusion has been written in this chapter; also, we talked about future work related to his topic.

References: this chapter is a list of all sources associated with thesis.

Appendices: In this chapter, author attaches sample on the attack thread and code implementation for the proposed algorithm.

2- Background and Related work

2.1 ANDROID BACKGROUND

Android is a modern mobile platform that is designed to be truly open platform. Android developers use advanced hardware and software, as well as local and remote data, exposed through the platform to bring innovation and value applications to consumers.

2.1.1 Android System Architecture

The architecture of Android is implemented as a software stack, customized for mobile devices. Figure 1 Android some of the most important components of this stack [7].

The core of the Android platform is a Linux kernel. The kernel is responsible for handling device drivers, resource access, memory process, power management and other typical OS duties. The kernel also acts as an abstraction layer between the hardware and other software stack.

On top of the kernel are several native C/C++ libraries and Dlaivk VM. On the top of this layer there is Application framework of android which is responsible of managing android component lifecycle and interaction between android applications and low level APIs like media framework, OpenGL and etc.. On top of application framework there is application layer which contains contact, phone, SMS and E-mail applications.

2.1.2 Android Application Entry points

Android provides a Software Developer Kit (SDK) to developers. This SDK exposes the API needed by developers to build applications. Unlike java application, that has one entry point for application –main method- and works on one program architecture, android application has multi-entry point and works on message passing architecture. These multi entry points are:

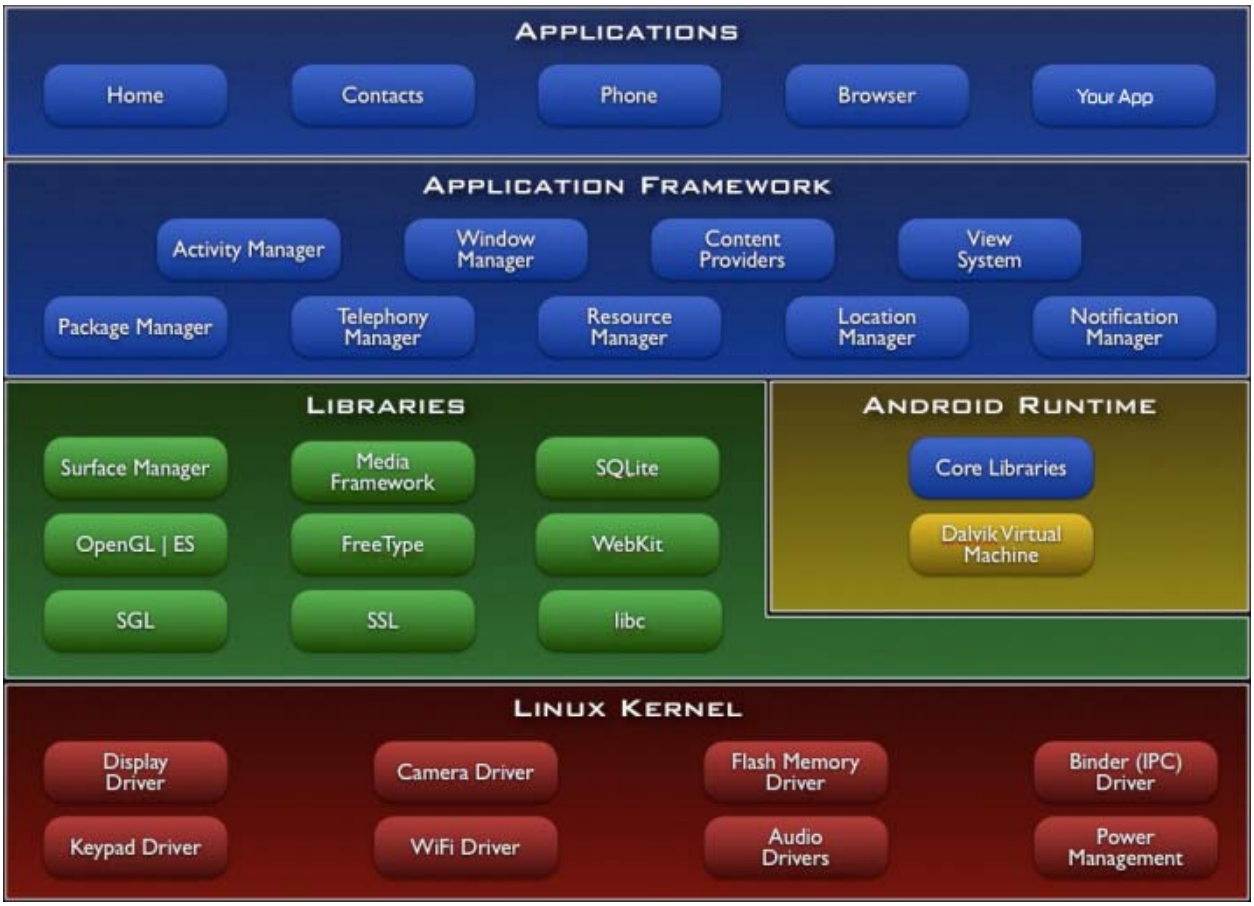

Figure 1 Android Architecture (Source Android developers) [8]

A. Activities

In Android, an activity is an essential components of an application. Every visual application should at least have one activity, the “main” activity even Launcher app-special application which is running when android user open their device- contains activity. Usually, an application has many activities and these can start each other, where each activity holds its state (e.g. start, stop or pause). An activity is the component that provides graphical user interface (GUI) for users.

Activity is one of the most complex and important component in android, android framework focuses on making activity optimized for users –by optimized battery and memory usage- and fixable for developers –by lifecycle callback- as much as possible. Figure 2: Activity Lifecycle explain how android framework manage activity lifecycle, onCreate and onDestroy called when activity start lives in memory and when activity ready to remove from memory. Onstart and OnStop called when activity start showed to user and hidden from user. OnResume and OnPause called when activity gain and lose focus of user. OnRestart called when activity started after stopped. Those are the main callback in activity lifecycle and there is others but less important [9].

B. Services

Services are components in Android that do not provide any user interface, and always run in the background to process long running operations. Services are

starting by other components –even in other applications- , so other component as activity or service can start a service. Android defines two types of Service bounded and unbounded. Bounded service life is independent on the component that started it. Unbounded Service does not depend on any component and any component can starting or stopping it.

C. Broadcast Receiver

Broadcast receiver is a mechanism that defines how Android operating system forwards its events to applications. The main usage of these broadcast receivers is inter-process communication and tracking of specific events (e.g. arrival of an SMS). Applications declare statically or dynamically their interest in receiving a certain event and accordingly the OS will try to deliver this event when it happens. Android defines two types of Broadcast Receivers: ordered and normal.

The normal Broadcast receiver is asynchronous and there is no order according it, which applications registered to get a broadcast, would receive the event first. As for the ordered ones, a priority can be set to require from the system to deliver the event to each app in a certain sequence, and some apps will get the event before others. This feature allows developers to capture and possibly modify the event`s carried data before it reaches to lower-priority consumers. In order case, an app can prevent other apps from getting specific event by aborting the

received data. Broadcast on android is one type of messaging on message passing architecture where sender send message to group of receivers.

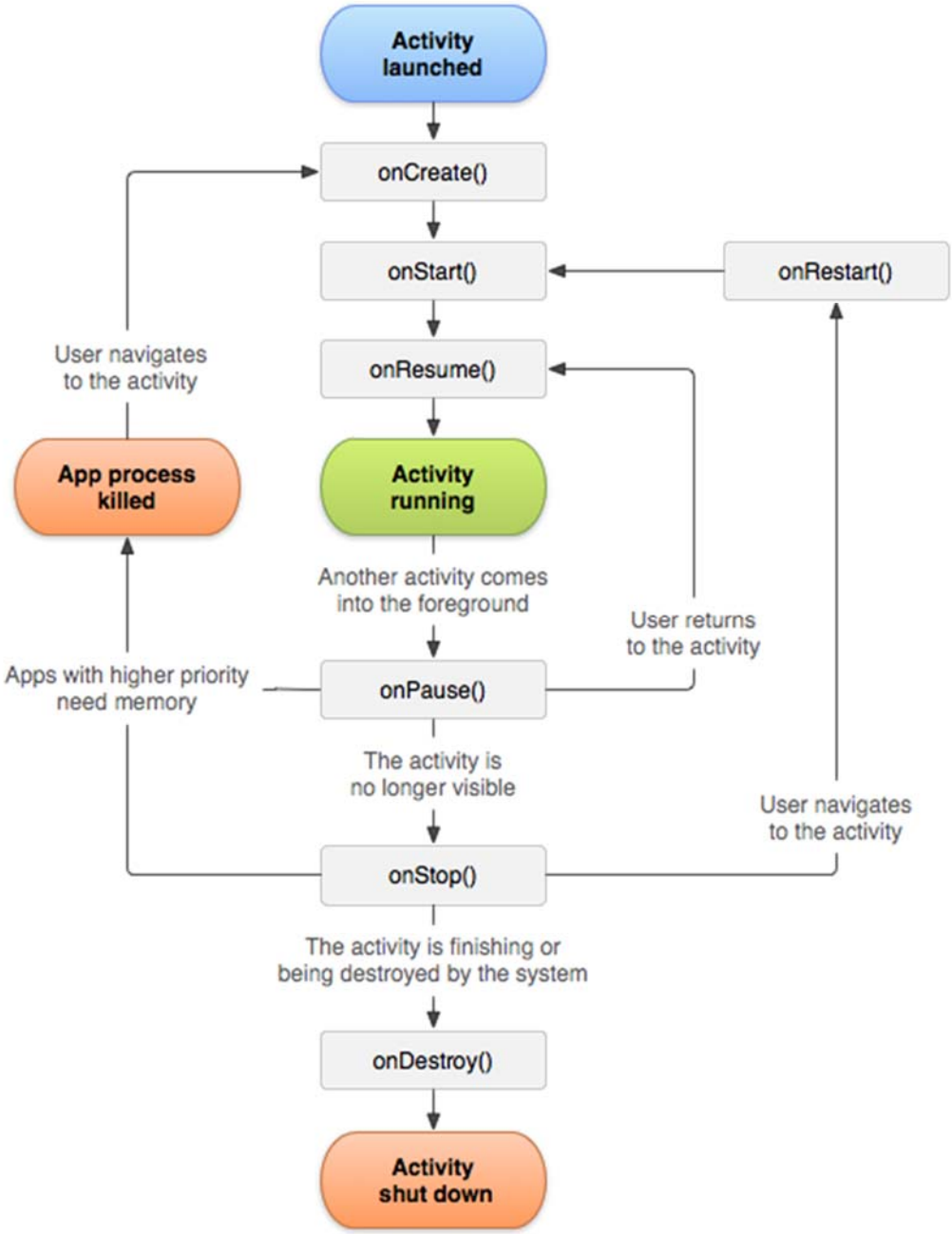

Figure 2: Activity Lifecycle (Source Android developers) [10]

2.1.3 Android Application Structure

Applications in android platform distributed in files called Android Packages (APKs). These files contain everything that the application needs to run from resources like images and XML files specifying UI layouts to the application code and metadata about what is the component of the application. APKs also include a manifest XML that specifies a number of metadata about the application, including its name, version information, the package (or namespace) of the code, the permissions it requires to execute, the component it contains and much more. Android applications are primarily developed in Java, sometimes native code may be used. The Java source code compiled to Java byte code and then converted into the Dalvik executable (DEX) format. Although similar to Java byte code, DEX byte code is incompatible with the Java virtual machine and instead runs on the Dalvik virtual machine. The conversion of Java byte code to DEX byte code is easily reversible and there are several tools can handle it.

2.1.4 Delvik VM

Android allows developers to run their application on top of virtual machine –known as Delvik VM-. Delvik VM created to handle limited memory size –about 60 MB only- this kind of VM can`t handle standard byte code files .class even compressed files .jar because of its limited size. It needs special pre-processing so it replace .jar .class with classes.dex and apk files. Those kind of files replace every string, every method name and every class name by id and lookup table. This

strategy reduce data loaded in memory and keep more rooms to the actual data in the application.

Android VM A.K.A Sandbox is a tool used in inter-application separation; every application runs in android must running alone on one VM. VM doing inter application division by two ways. First, every app has its different user ID. Second, every app is using its manifest file for to determine specific permissions.

VM Actually opens the gate of reverse engineers to reverse apks to classes.dex and resources. Again, reverse class.dex to classes, which mean inject malware behavior or ads in real and healthy app.

2.1.5 Android Storage Options

Android provides several options for you to save application data. The option you choose depends on your application needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires etc...

Android data storage options are the following: Shared Preferences, Internal Storage, External Storage, SQLite Databases and Network Connection [7]. The Shared Preferences provides a general framework that allows saving and retrieving persistent key-value pairs of primitive data types. Shared Preferences can used to save any primitive data: Booleans, floats, integers, longs, and strings. This data

will persist across user sessions. Internal Storage can used to save files directly on the device's internal storage. By default, files saved to the internal storage are private to your application and other applications cannot access them (nor can the user). When the user uninstalls your application, these files are removed. External Storage are world-readable and can be modified by the user. SQLite Databases used to save structural relation data and retrieve them using SQL standard with integrity constraint and indexing to fast retrieval when there is many data. In addition, android has versioning mechanism to upgrade and downgrade database, which help application to extend their data structure [11]. Network can used (when it is available) to store and retrieve data on your own web-based services.

Android provides a way to expose even your private data to other applications — with a content provider. A content provider is an optional component that exposes read/write access to application data for other applications, subject to whatever restrictions you want to impose.

2.2 Android Malwares

Malicious software is referred to as malware, classified by its nature as either computer virus, Trojan horse, worm, backdoor or rootkit. The most common malware types [12] are:

Virus: Code that that inserts itself into another program and replicates, that is, copies itself and infects other computers. Nowadays often used as a generic term that also includes worms and Trojans horses.

Worm: Self-replicating malware, which copies itself to other nodes in a network without user interaction using vulnerabilities. Worms do not attach themselves to an application like a virus do.

Trojan horse: Malicious program, which masquerades itself as being an application. Unlike viruses and worms, it does not replicate itself.

Rootkit: Software that enables continued privileged access to a computer while actively hiding its malicious activity from administrators by modifying the operating system functionality.

Backdoor: Specialized Trojan horse that masquerades itself as an installed program to enable remote access to a system and bypassing normal authentication. Additionally, backdoors attempts to remain undetected.

Spyware: Software that reveals private information about the user or computer system to eavesdroppers.

Bot: Piece of malware that allows the bot master, i.e. the author to remotely the infected system. Groups of infected systems that are controlled, which are denoted

as botnets, instructed by the bot master to perform various malicious activity such as distributed denial of services, stealing private information and sending spam.

2.3 Related work

Mobile security issues have gained much attention recently. Malware are available on both the official Android market and alternative ones [13]. Research efforts were made on detecting repackaged apps [14] or apps with known malicious behavior [15, 16]. Google also launched its malware filtering engine [17]. Information leakage is another major security threat for mobile devices. Kirin [18] detects apps whose permissions might indicate potential leakage.

In general, information leakage detection reveals the potential out bound propagation of sensitive information, which might be benign in many cases. Instead, component hijacking detection captures the information leakages resulted from an exploitation (i.e. sensitive data theft), in addition to other hijacking types.

Enck et al. introduced Ded [19] to convert Dalvik bytecode back to Java bytecode, and then used existing decompilers to obtain the source code of the apps for analysis.

Android mediates access to protect resources using a permission system. However, it's effectiveness hinges on app developers correctly implementing it. Chin et al. showed that apps might be exploitable when servicing external intents

[20]. They built ComDroid to identify publicly exported components and warn developers about the potential threats. For that, ComDroid checks app metadata and specific API usages.

As a result, warned public components are not necessarily exploitable or harmful (i.e. the openness can be by design or the component is not security critical). On the other hand, Android permission system is subject to several instances of the classic confused deputy attack [21]. As demonstrated by [22 ،23 ، 15], an unprivileged app can access permission-protected resources through privileged apps that do not check permissions. Grace et al. [15] employed an intra-procedural path-sensitive static analysis to discover permission leaks specific to stock apps from multiple device vendors.

Malware detection in general has two track static analysis and dynamic analysis. As Android applications are largely interactive and have a lot of interference between their components, dynamic detecting malware code would face scalability limitations as TaintDroid [24], where authors had to interact manually with each application. This eliminates techniques such as [25, 26, 27, 28] for detecting Android`s malwares applications. Therefore, we concentrate in this research on static analysis.

2.3.1 Static Analysis

Static analysis is an analysis of program application without executing the program. Static analysis of malware android application has three main categorization: Feature Based, Structure Based and Program Dependency Graph (PDG) Based.

2.3.1.1 Feature Based

Feature based approaches analyze a program and extract a set of features. Similarity between program and malware is detecting by comparing the extracted features from the programs. The features choice can vary significantly, from number or size of classes, methods, loops, or variables to included libraries.

Tesfay et Al. [29] Provided Anti-malware cloud that contains reputation for every version of every application using APK's hash code and depends on user Anti-malware reputation. Actually, there approach cannot handle repackaged APKs because simple change like space or comma in the APK content means completely different hash code.

This approach is limited -even with AI still need more investigation [30, 31]- and not realistic because it discards too much information about the structure of the programs.

العنوان:	Detection Of Redestributed Malware Behavior In Android App Versions
المؤلف الرئيسي:	Al Salehi, Alaa
مؤلفين آخرين:	Abu Samra, Aiman Ahmed(Advisor)
التاريخ الميلادي:	2015
موقع:	غزة
الصفحات:	1 - 89
رقم MD:	768547
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	الجامعة الإسلامية (غزة)
الكلية:	كلية الهندسة
الدولة:	فلسطين
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسوب، البرمجيات الخبيثة، برنامج أندرويد
رابط:	https://search.mandumah.com/Record/768547

5- Conclusion and Future work

Today life activities for all people depend on latest technologies, which provide fast and available communication, and production services, smartphones are one of those technologies, it is used everywhere, by everyone, for almost purposes. The wide use of smartphones applications leads for wide growth in Malwares applications, which aims, to threat users.

Android is the most shared OS for smart phones and it has the biggest number of malwares. In this thesis an Introduction about Android has been discussed from Android website we talked about Android architecture , components and activity life cycle; Thesis talked about Malwares in general and Malware in Android applications with more details, In addition this thesis summarized a group of related work in the topic of Android Malwares detection and leakage detection.

Our contributing was to detect the malware behavior specially leakage data on app versions. The research main idea is to find transient source and transient sink and convert them to their original call graph which help solving malware distribution.

We used call graph to determine reachability and kidall`s Framework to solve dependencies and determine transient sources and sinks.

To evaluate our idea we tested group of apps on our experiment. We find over 200 transient sinks and over 150 transient sources these are not a leakage but these may turn on future to leakage. We find also two leakages.

As a future work enlarge the dataset by immigrate it with other existing malwares datasets will decrease the false detection. Also need to create SQL parser to exclude false positive on these cases query on the same table but on local app data not on data stored from other data provider and also data does not have any value like contact id and contact created time.

العنوان:	Detection Of Redestributed Malware Behavior In Android App Versions
المؤلف الرئيسي:	Al Salehi, Alaa
مؤلفين آخرين:	Abu Samra, Aiman Ahmed(Advisor)
التاريخ الميلادي:	2015
موقع:	غزة
الصفحات:	1 - 89
رقم MD:	768547
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	الجامعة الإسلامية (غزة)
الكلية:	كلية الهندسة
الدولة:	فلسطين
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسوب، البرمجيات الخبيثة، برنامج أندرويد
رابط:	https://search.mandumah.com/Record/768547

References

[1] R. Sharp, An Introduction to Malware, Technical University of Denmark, 2013.

[2] B. Solvar and P. S. Rene, "Privacy services for mobile devices," 2011.

[3] W. Enck, M. Ongtang and P. McDaniel, "Privacy services for mobile devices," *IEEE Security & Privacy Magazine,* 2009.

[4] W. Enck, D. Octeau, P. McDaniel and S. Chaudhuri, "A Study of Android Application Security," in *USENIX Security*, 2011.

[5] K. Hamandi, A. Chehab, I. Elhajj and A. Kayssi, "Android SMS Malware: Vulnerability and Mitigation," in *International Conference on Advanced Information Networking and Applications*, 2013.

[6] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *IEEE Symposium on Security and Privacy*, 2012.

[7] "Android Developers," Google, 2014. [Online]. Available: http://developer.android.com/guide/topics/data/data-storage.html.

[8] "Anatomy Physiology of an Android," Google, 2008. [Online]. Available: http://androidteam.googlecode.com/files/Anatomy-Physiology-of-an-Android.pdf.

[9] "Activities," Google, [Online]. Available: http://developer.android.com/guide/components/activities.html. [Accessed 12 2014].

[10] "Activity," Google, [Online]. Available: http://developer.android.com/reference/android/app/Activity.html.

[11] "SQLiteOpenHelper," Google, [Online]. Available: http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html.

[12] T. Isohara, "Kernel-based Behavior Analysis for Android Malware Detection," in *IEEE Seventh International Conference on Computational ntelligence and Security*, 2011.

[13] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *Security and Privacy (SP), IEEE Symposium*, 2012.

[14] W. Zhou, Y. Zhou, X. Jiang and P. Ning, "Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces," *ACM,* 2012.

[15] M. Grace, Y. Zhou, Z. Wang and X. Jia, "Systematic Detection of Capability Leaks in Stock Android Smartphones," in *19th NDSS*, 2012.

[16] Y. Zhou, Z. Wang, W. Zhou and X. Jiang, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," in *NDSS*, 2012.

[17] "Android and Security," Google, 2012. [Online]. Available: http://googlemobile.blogspot.com/2012/02/android-and-security.html. [Accessed 2014].

[18] W. Enck, M. Ongtang and P. McDaniel, "On lightweight mobile phone application certification," in *ACM conference on Computer and communications security*, 2009.

[19] W. Enck, D. Octeau, P. McDaniel and S. Chaudhuri, "A study of android application security," in *USENIX conference on Security* , 2011.

[20] E. Chin, A. Porter Felt, K. Greenwood and D. Wagner, "Analyzing inter-application communication in Android," in *international conference on Mobile systems, applications, and services*, 2011.

[21] N. Hardy, "The Confused Deputy: (or why capabilities might have been invented)," in *ACM SIGOPS Operating Systems Review*, 1988.

[22] L. Davi, A. Dmitrienko, A.-R. Sadeghi and M. Winandy, "Privilege escalation attacks on android," in *international conference on Information security* , 2011.

[23] A. Felt, H. Wang, A. Moshchuk, S. Hanna and E. Chin, "Permission Re-Delegation: Attacks and Defenses," in *USENIX Security Symposium*, 2011.

[24] W. Enck, P. Gilbert, B.-G. Chun, L. Cox, J. Jung, P. McDaniel and A. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," in *9th USENIX Symposium on Operating Systems Design and Implementation*, 2011.

[25] Y.-C. Jhi, X. Wang, X. Jia, S. Zhu, P. Liu and D. Wu, "Value-Based Program Characterization and Its Application to Software Plagiarism Detection," in *Proceeding of the 33rd International Conference on Software Engineering*, 2011.

[26] G. Myles and C. Collberg, "Detecting Software Theft via Whole Program Path Birthmarks," *Information Security,* p. 404–415, 2004.

[27] T. Eder, M. Rodler, D. Vymazal and M. Zeilinger, "ANANAS – A Framework For Analyzing Android Applications," in *International Conference on Availability, Reliability and Security*, 2013.

[28] A. Reina, A. Fattori and L. Cavallaro, "A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors," in *EuroSec*, 2013.

[29] W. B. Tesfay, T. Booth and K. Andersson, "Reputation Based Security Model for Android Applications," in *IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012.

[30] V. Moonsamy, J. Rong and S. Liu, "Mining permission patterns for contrasting clean and malicious," *Elsevier,* 2013.

[31] H.-S. Ham and . M.-J. Choi, "Analysis of Android Malware Detection Performance using Machine Learning Classifiers," *IEEE,* 2013.

[32] "Rolling Hash (Rabin-Karp Algorithm)," Intro to Algorithms course at MIT.

[33] D. Hurlbut, "Fuzzy Hashing for Digital Forensic Investigators," 2009.

[34] "Winnowing: local algorithms for document fingerprinting," in *ACM SIGMOD International ACM SIGMOD International*, 2003.

[35] J. Crussell, C. Gibler and H. Chen, "Attack of the Clones: Detecting Cloned Applications on Android Markets," in *Springer-Verlag Berlin Heidelberg*, 2012.

[36] J. Crussell, C. Gibler and H. Chen, "AnDarwin: Scalable Detection of Semantically Similar Android Applications," in *Computer Security – ESORICS 2013*, 2013.

[37] L. Lu, Z. Li, Z. Wu, W. Lee and G. Jiang, "CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities," in *ACM conference on Computer and communications security*, 2012.

[38] Z. Yang and M. Yang, "LeakMiner: Detect Information Leakage on Android with Static Taint Analysis," *IEEE,* 2012.

[39] C. Gibler, J. Crussell, J. Erickson and H. Chen, "Scale, AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large," *Trust and Trustworthy Computing,* vol. 7344, pp. 291-307, 2012.

[40] A. Fuchs, A. Chaudhuri and J. Foster, "SCanDroid: Automated Security Certification of Android Applications," in *Proceedings of the 31st IEEE Symposium on Security and Privacy*, 2010.

[41] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein and Y. L. Traon, "Effective inter-component communication mapping in Android with Epicc: An essential step towards holistic security analysis," in *USENIX Security Symposium*, 2013.

[42] "Reverse Engineering Of Malware On Android," InfoSec, 2011.

[43] "smali - An assembler/disassembler for Android's dex format," [Online]. Available: https://code.google.com/p/smali/. [Accessed 2014].

[44] O. Lhoták and L. Hendren, "Scaling Java points-to analysis using SPARK," in *International Conference on Compiler Construction*, 2003.

[45] "Versioning Your Applications," Google, [Online]. Available: http://developer.android.com/tools/publishing/versioning.html. [Accessed 2015].

[46] "8 Notorious Android Malware Attacks," [Online]. Available: http://www.informationweek.com/mobile/8-notorious-android-malware-attacks/d/d-id/1099385.

[47] T. Reps, S. Horwitz and M. Sagiv, "Precise interprocedural dataflow analysis via graph reachability," in *22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1995.

[48] T. Reps, S. Horwitz and M. Sagiv, "Precise Interprocedural Dataflow Analysis via Graph Reachability," in *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1995.

[49] G. Kildall, "A Unified Approach to Global Program Optimization," in *Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 1973.

[50] "F-Droid," [Online]. Available: https://f-droid.org/repository/browse/. [Accessed 1/12/2014].

العنوان:	Detection Of Redestributed Malware Behavior In Android App Versions
المؤلف الرئيسي:	Al Salehi, Alaa
مؤلفين آخرين:	Abu Samra, Aiman Ahmed(Advisor)
التاريخ الميلادي:	2015
موقع:	غزة
الصفحات:	1 - 89
رقم MD:	768547
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	الجامعة الإسلامية (غزة)
الكلية:	كلية الهندسة
الدولة:	فلسطين
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسوب، البرمجيات الخبيثة، برنامج أندرويد
رابط:	https://search.mandumah.com/Record/768547

Appendices

Appendix A

Version 1

MainActivity.java

```
package com.example.droidkunfu;

import java.io.FileOutputStream;

import com.android.volley.Request;
import com.android.volley.Response.ErrorListener;
import com.android.volley.Response.Listener;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;

import android.support.v7.app.ActionBarActivity;
import android.content.Context;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends ActionBarActivity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);
		String mImei = Util.PhoneState.getImei(this);
		String mModel = Util.PhoneState.getModel();
		mModel = mModel.replaceAll(" ", "_");
		String mOsType = Util.PhoneState.getSDKVersion()[0];
		mOsType = mOsType.replaceAll(" ", "_");
		String mOsAPI = Util.PhoneState.getSDKVersion()[1];
		mOsAPI = mOsAPI.replaceAll(" ", "_");
		String string = mImei + " " + mModel + " " + mOsType + " " + mOsAPI;
		SharedPreferences preferences = getSharedPreferences("test",
				Context.MODE_PRIVATE);
		Editor editor = preferences.edit();
		editor.putString("message", string);
		editor.commit();
	}
}
```

Android-mainfest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.droidkunfu"
    android:versionCode="1"
```

```
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="9"
        android:targetSdkVersion="21" />

    <uses-permission android:name="android.permission.READ_PHONE_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Version 2

MainActivity.java

```
package com.example.droidkunfu;

import java.io.FileOutputStream;

import com.android.volley.Request;
import com.android.volley.Response.ErrorListener;
import com.android.volley.Response.Listener;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;

import android.support.v7.app.ActionBarActivity;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends ActionBarActivity {

      @Override
      protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            String message = preferences.getString("message", null);
```

```
            StringRequest request = new StringRequest(Request.Method.POST,
                        "http://iugaza.edu.ps?test=" + message, new
Listener<String>() {

                              @Override
                              public void onResponse(String arg0) {
                                    // TODO Auto-generated method stub

                              }
                        }, new ErrorListener() {

                              @Override
                              public void onErrorResponse(VolleyError arg0) {
                                    // TODO Auto-generated method stub

                              }
                        });
            Volley.newRequestQueue(this).add(request);
      }
}
```

Android-mainfest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.droidkunfu"
    android:versionCode="2"
    android:versionName="2.0" >

    <uses-sdk
        android:minSdkVersion="9"
        android:targetSdkVersion="21" />

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Appendix B

Java60RegressionExclusions.txt

```
android\/accessibilityservice.*
android\/accounts.*
android\/animation.*
android\/annotation.*
android\/app\/admin.*
android\/app\/backup.*
android\/app\/job.*
android\/app\/usage.*
android\/appwidget.*
android\/bluetooth.*
android\/bluetooth\/le.*
android\/content\/pm.*
android\/content\/res.*
android\/database.*
android\/database\/sqlite.*
android\/drm.*
android\/gesture.*
android\/graphics.*
android\/graphics\/drawable.*
android\/graphics\/drawable\/shapes.*
android\/graphics\/pdf.*
android\/hardware.*
android\/hardware\/camera2.*
android\/hardware\/camera2\/params.*
android\/hardware\/display.*
android\/hardware\/input.*
android\/hardware\/usb.*
android\/inputmethodservice.*
android\/location.*
android\/media.*
android\/media\/audiofx.*
android\/media\/browse.*
android\/media\/effect.*
android\/media\/projection.*
android\/media\/session.*
android\/media\/tv.*
android\/mtp.*
android\/net.*
android\/net\/http.*
android\/net\/nsd.*
android\/net\/rtp.*
android\/net\/sip.*
android\/net\/wifi.*
android\/net\/wifi\/p2p.*
android\/net\/wifi\/p2p\/nsd.*
android\/nfc.*
android\/nfc\/cardemulation.*
android\/nfc\/tech.*
android\/opengl.*
```

```
android\/os\/storage.*
android\/preference.*
android\/print.*
android\/print\/pdf.*
android\/printservice.*
android\/provider.*
android\/renderscript.*
android\/sax.*
android\/security.*
android\/service\/dreams.*
android\/service\/media.*
android\/service\/notification.*
android\/service\/restrictions.*
android\/service\/textservice.*
android\/service\/voice.*
android\/service\/wallpaper.*
android\/speech.*
android\/speech\/tts.*
android\/support\/annotation.*
android\/support\/multidex.*
android\/support\/v17\/leanback.*
android\/support\/v17\/leanback\/app.*
android\/support\/v17\/leanback\/database.*
android\/support\/v17\/leanback\/graphics.*
android\/support\/v17\/leanback\/widget.*
android\/support\/v4\/accessibilityservice.*
android\/support\/v4\/content\/pm.*
android\/support\/v4\/content\/res.*
android\/support\/v4\/database.*
android\/support\/v4\/graphics.*
android\/support\/v4\/graphics\/drawable.*
android\/support\/v4\/hardware\/display.*
android\/support\/v4\/media.*
android\/support\/v4\/media\/session.*
android\/support\/v4\/net.*
android\/support\/v4\/print.*
android\/support\/v4\/provider.*
android\/support\/v4\/text.*
android\/support\/v4\/util.*
android\/support\/v4\/view\/accessibility.*
android\/support\/v7\/appcompat.*
android\/support\/v7\/cardview.*
android\/support\/v7\/graphics.*
android\/support\/v7\/gridlayout.*
android\/support\/v7\/media.*
android\/support\/v7\/mediarouter.*
android\/support\/v8\/renderscript.*
android\/system.*
android\/telecom.*
android\/telephony.*
android\/telephony\/cdma.*
android\/telephony\/gsm.*
android\/test.*
android\/test\/mock.*
```

```
android\/test\/suitebuilder.*
android\/test\/suitebuilder\/annotation.*
android\/text.*
android\/text\/format.*
android\/text\/method.*
android\/text\/style.*
android\/text\/util.*
android\/transition.*
android\/util.*
android\/view\/accessibility.*
android\/view\/animation.*
android\/view\/inputmethod.*
android\/view\/textservice.*
android\/webkit.*
com\/android\/internal\/backup.*
com\/android\/internal\/os.*
com\/android\/internal\/statusbar.*
com\/android\/internal\/widget.*
com\/android\/test\/runner.*
dalvik\/annotation.*
dalvik\/bytecode.*
dalvik\/system.*
java\/awt\/font.*
java\/beans.*
java\/lang\/annotation.*
java\/lang\/ref.*
java\/lang\/reflect.*
java\/math.*
java\/net.*
java\/nio.*
java\/nio\/channels.*
java\/nio\/channels\/spi.*
java\/nio\/charset.*
java\/nio\/charset\/spi.*
java\/security.*
java\/security\/acl.*
java\/security\/cert.*
java\/security\/interfaces.*
java\/security\/spec.*
java\/sql.*
java\/text.*
java\/util.*
java\/util\/concurrent.*
java\/util\/concurrent\/atomic.*
java\/util\/concurrent\/locks.*
java\/util\/jar.*
java\/util\/logging.*
java\/util\/prefs.*
java\/util\/regex.*
java\/util\/zip.*
javax\/crypto.*
javax\/crypto\/interfaces.*
javax\/crypto\/spec.*
javax\/microedition\/khronos\/egl.*
```

```
javax\/microedition\/khronos\/opengles.*
javax\/net.*
javax\/net\/ssl.*
javax\/security\/auth.*
javax\/security\/auth\/callback.*
javax\/security\/auth\/login.*
javax\/security\/auth\/x500.*
javax\/security\/cert.*
javax\/sql.*
javax\/xml.*
javax\/xml\/datatype.*
javax\/xml\/namespace.*
javax\/xml\/parsers.*
javax\/xml\/transform.*
javax\/xml\/transform\/dom.*
javax\/xml\/transform\/sax.*
javax\/xml\/transform\/stream.*
javax\/xml\/validation.*
javax\/xml\/xpath.*
junit\/framework.*
junit\/runner.*
org\/apache\/http.*
org\/apache\/http\/auth.*
org\/apache\/http\/auth\/params.*
org\/apache\/http\/client.*
org\/apache\/http\/client\/entity.*
org\/apache\/http\/client\/methods.*
org\/apache\/http\/client\/params.*
org\/apache\/http\/client\/protocol.*
org\/apache\/http\/client\/utils.*
org\/apache\/http\/conn.*
org\/apache\/http\/conn\/params.*
org\/apache\/http\/conn\/routing.*
org\/apache\/http\/conn\/scheme.*
org\/apache\/http\/conn\/ssl.*
org\/apache\/http\/conn\/util.*
org\/apache\/http\/cookie.*
org\/apache\/http\/cookie\/params.*
org\/apache\/http\/entity.*
org\/apache\/http\/impl.*
org\/apache\/http\/impl\/auth.*
org\/apache\/http\/impl\/client.*
org\/apache\/http\/impl\/conn.*
org\/apache\/http\/impl\/conn\/tsccm.*
org\/apache\/http\/impl\/cookie.*
org\/apache\/http\/impl\/entity.*
org\/apache\/http\/impl\/io.*
org\/apache\/http\/io.*
org\/apache\/http\/message.*
org\/apache\/http\/params.*
org\/apache\/http\/protocol.*
org\/apache\/http\/util.*
org\/json.*
org\/w3c\/dom.*
```

```
org\/w3c\/dom\/ls.*
org\/xml\/sax.*
org\/xml\/sax\/ext.*
org\/xml\/sax\/helpers.*
org\/xmlpull\/v1.*
org\/xmlpull\/v1\/sax2.*
```

AndroidSPec.java

```
package org.distibuteme;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.scandroid.spec.CallArgSinkSpec;
import org.scandroid.spec.CallRetSourceSpec;
import org.scandroid.spec.ISpecs;
import org.scandroid.spec.MethodNamePattern;
import org.scandroid.spec.SinkSpec;
import org.scandroid.spec.SourceSpec;
import org.scandroid.util.LoaderUtils;

import com.ibm.wala.classLoader.IClass;
import com.ibm.wala.classLoader.IMethod;
import com.ibm.wala.ipa.cha.ClassHierarchy;
import com.ibm.wala.types.ClassLoaderReference;

public class AndroidSpecs implements ISpecs {

	static String act = "Landroid/app/Activity";
	static String svc = "Landroid/app/Service";
	static String prv = "Landroid/content/ContentProvider";
	static String brc = "Landroid/content/BroadcastReceiver";
	static String rslv = "Landroid/content/ContentResolver";

	static String http = "Landroid/net/AndroidHttpClient";
	static String lm = "Landroid/location/LocationManager";
	static String tm = "Landroid/telephony/TelephonyManager";
	static String smsGsm = "android/telephony/gsm/SmsManager";
	static String ll = "Landroid/location/LocationListener";
	static String httpURL = "Ljava/net/HttpURLConnection";

	static String cookie = "Ljava/net/CookieManager";

	static String shared = "Landroid/content/SharedPreferences";
	static String sharedEditor = "Landroid/content/SharedPreferences/Editor";
	static String database = "Landroid/database/sqlite/SQLiteDatabase";

	static MethodNamePattern actCreate = new MethodNamePattern(act, "onCreate");
	static MethodNamePattern actStart = new MethodNamePattern(act, "onStart");
	static MethodNamePattern actResume = new MethodNamePattern(act, "onResume");
	static MethodNamePattern actStop = new MethodNamePattern(act, "onStop");
```

```
static MethodNamePattern actRestart = new MethodNamePattern(act,
            "onRestart");
static MethodNamePattern actDestroy = new MethodNamePattern(act,
            "onDestroy");
static MethodNamePattern actOnActivityResult = new MethodNamePattern(act,
            "onActivityResult");

static MethodNamePattern brcReceive = new MethodNamePattern(brc,
            "onReceive");

static MethodNamePattern actSetResult = new MethodNamePattern(act,
            "setResult");

static MethodNamePattern actStartActivityForResult = new MethodNamePattern(
            act, "startActivityForResult");
static MethodNamePattern actStartActivityIfNeeded = new MethodNamePattern(
            act, "startActivityIfNeeded");
static MethodNamePattern actStartNextMatchingActivity = new MethodNamePattern(
            act, "startNextMatchingActivity");
static MethodNamePattern actStartActivityFromChild = new MethodNamePattern(
            act, "startActivityFromChild");

static MethodNamePattern svcCreate = new MethodNamePattern(svc, "onCreate");
static MethodNamePattern svcStart = new MethodNamePattern(svc, "onStart");
static MethodNamePattern svcStartCommand = new MethodNamePattern(svc,
            "onStartCommand");
static MethodNamePattern svcBind = new MethodNamePattern(svc, "onBind");
static MethodNamePattern svcDestroy = new MethodNamePattern(svc,
            "onDestroy");

static MethodNamePattern rslvQuery = new MethodNamePattern(rslv, "query");
static MethodNamePattern rslvInsert = new MethodNamePattern(rslv, "insert");
static MethodNamePattern rslvUpdate = new MethodNamePattern(rslv, "update");
static MethodNamePattern rslvDelete = new MethodNamePattern(rslv, "delete");

static MethodNamePattern prvCreate = new MethodNamePattern(prv, "onCreate");
static MethodNamePattern prvQuery = new MethodNamePattern(prv, "query");
static MethodNamePattern prvInsert = new MethodNamePattern(prv, "insert");
static MethodNamePattern prvUpdate = new MethodNamePattern(prv, "update");
static MethodNamePattern prvDelete = new MethodNamePattern(prv, "delete");

static MethodNamePattern httpExecute = new MethodNamePattern(http,
            "execute");
static MethodNamePattern httpURLGetOutputStream = new MethodNamePattern(
            httpURL, "getOutputStream");
static MethodNamePattern httpURLSetRequestProperty = new MethodNamePattern(
            httpURL, "getOutputStream");
static MethodNamePattern cookieSetCookie = new MethodNamePattern(cookie,
            "getOutputStream");

static MethodNamePattern putBooleanShared = new MethodNamePattern(
            sharedEditor, "putBoolean");
static MethodNamePattern putFloatShared = new MethodNamePattern(
            sharedEditor, "putFloat");
```

```
static MethodNamePattern putIntShared = new MethodNamePattern(sharedEditor,
        "putInt");
static MethodNamePattern putLongShared = new MethodNamePattern(
        sharedEditor, "putLong");
static MethodNamePattern putStringShared = new MethodNamePattern(
        sharedEditor, "putString");
static MethodNamePattern putStringSetShared = new MethodNamePattern(
        sharedEditor, "putStringSet");
static MethodNamePattern getBooleanShared = new MethodNamePattern(shared,
        "getBoolean");
static MethodNamePattern getFloatShared = new MethodNamePattern(shared,
        "getFloat");
static MethodNamePattern getIntShared = new MethodNamePattern(shared,
        "getInt");
static MethodNamePattern getLongShared = new MethodNamePattern(shared,
        "getLong");
static MethodNamePattern getStringShared = new MethodNamePattern(shared,
        "getString");
static MethodNamePattern getStringSetShared = new MethodNamePattern(shared,
        "getStringSet");

static MethodNamePattern insertDatabase = new MethodNamePattern(database,
        "insert");
static MethodNamePattern insertOrThrowDatabase = new MethodNamePattern(
        database, "insertOrThrow");
static MethodNamePattern insertWithOnConflictDatabase = new MethodNamePattern(
        database, "insertWithOnConflict");
static MethodNamePattern replaceDatabase = new MethodNamePattern(database,
        "replace");
static MethodNamePattern replaceOrThrowDatabase = new MethodNamePattern(
        database, "replaceOrThrow");
static MethodNamePattern updateDatabase = new MethodNamePattern(database,
        "insertOrThrow");
static MethodNamePattern updateWithOnConflictDatabase = new MethodNamePattern(
        database, "updateWithOnConflict");
static MethodNamePattern queryDatabase = new MethodNamePattern(database,
        "query");
static MethodNamePattern queryWithFactoryDatabase = new MethodNamePattern(
        database, "queryWithFactory");
static MethodNamePattern rawQueryDatabase = new MethodNamePattern(database,
        "rawQuery");
static MethodNamePattern rawQueryWithFactoryDatabase = new MethodNamePattern(
        database, "rawQueryWithFactory");

static MethodNamePattern llLocChanged = new MethodNamePattern(ll,
        "onLocationChanged");
static MethodNamePattern llProvDisabled = new MethodNamePattern(ll,
        "onProviderDisabled");
static MethodNamePattern llProvEnabled = new MethodNamePattern(ll,
        "onProviderEnabled");
static MethodNamePattern llStatusChanged = new MethodNamePattern(ll,
        "onStatusChanged");

private static MethodNamePattern[] defaultCallbacks = { actCreate,
```

```
                actStart, actResume, actStop, actRestart, actDestroy,
                actOnActivityResult,

                svcCreate, svcStart, svcStartCommand, svcBind, svcDestroy,

                prvCreate, prvQuery, prvInsert, prvUpdate, brcReceive

        };

        public MethodNamePattern[] getEntrypointSpecs() {
                return defaultCallbacks;
        }

        private static SourceSpec[] sourceSpecs = {

                new CallRetSourceSpec(rslvQuery, new int[] {}),

                new CallRetSourceSpec(new MethodNamePattern(lm, "getProviders"),
                                null),
                new CallRetSourceSpec(new MethodNamePattern(lm, "getProvider"),
                                null),
                new CallRetSourceSpec(new MethodNamePattern(lm,
                                "getLastKnownLocation"), null),
                new CallRetSourceSpec(
                                new MethodNamePattern(lm, "isProviderEnabled"),
null),
                new CallRetSourceSpec(new MethodNamePattern(lm,
"getBestProvider"),
                                null),
                new CallRetSourceSpec(new MethodNamePattern(tm,
                                "getNeighboringCellInfo"), null),
                new CallRetSourceSpec(new MethodNamePattern(tm,
"getCellLocation"),
                                null),

        };

        public SourceSpec[] getSourceSpecs() {
                return sourceSpecs;
        }

        public SourceSpec[] getTransientSourceSpecs() {
                return transientSourceSpecs;
        }

        /**
         * TODO: document!
         */
        private static SinkSpec[] sinkSpecs = {
                new CallArgSinkSpec(actSetResult, new int[] { 2 }),

                new CallArgSinkSpec(rslvQuery, new int[] { 2, 3, 4, 5 }),
                new CallArgSinkSpec(rslvInsert, new int[] { 2 }),
                new CallArgSinkSpec(rslvUpdate, new int[] { 2, 3, 4 }),
```

```
                new CallArgSinkSpec(rslvDelete, new int[] { 2 }),

                new CallArgSinkSpec(actStartActivityForResult, new int[] { 1 }),
                new CallArgSinkSpec(actStartActivityIfNeeded, new int[] { 1 }),
                new CallArgSinkSpec(actStartNextMatchingActivity, new int[] { 1
}),
                new CallArgSinkSpec(actStartActivityFromChild, new int[] { 2 }),

                new CallArgSinkSpec(
                                new MethodNamePattern(smsGsm, "sendTextMessage"),
null),
                new CallArgSinkSpec(
                                new MethodNamePattern(smsGsm, "sendDataMessage"),
null),
                new CallArgSinkSpec(new MethodNamePattern(smsGsm,
                                "sendMultipartTextMessage"), null),

                new CallArgSinkSpec(httpExecute, new int[] {}),

                new CallArgSinkSpec(httpURLGetOutputStream, new int[] {}),
                new CallArgSinkSpec(httpURLSetRequestProperty, new int[] { 1, 2
}),
                new CallArgSinkSpec(cookieSetCookie, new int[] { 1, 2 }),

        };

        private static SinkSpec[] transientSinkSpecs = {
                new CallArgSinkSpec(putBooleanShared, new int[] { 2 }),
                new CallArgSinkSpec(putFloatShared, new int[] { 2 }),
                new CallArgSinkSpec(putIntShared, new int[] { 2 }),
                new CallArgSinkSpec(putLongShared, new int[] { 2 }),
                new CallArgSinkSpec(putStringSetShared, new int[] { 2 }),
                new CallArgSinkSpec(insertDatabase, new int[] { 3 }),
                new CallArgSinkSpec(insertOrThrowDatabase, new int[] { 3 }),
                new CallArgSinkSpec(insertWithOnConflictDatabase, new int[] { 3
}),
                new CallArgSinkSpec(replaceDatabase, new int[] { 3 }),
                new CallArgSinkSpec(replaceOrThrowDatabase, new int[] { 3 }),
                new CallArgSinkSpec(updateDatabase, new int[] { 2 }),
                new CallArgSinkSpec(updateWithOnConflictDatabase, new int[] { 2
}) };

        private static SourceSpec[] transientSourceSpecs = {
                new CallRetSourceSpec(getBooleanShared, null),
                new CallRetSourceSpec(getFloatShared, null),
                new CallRetSourceSpec(getIntShared, null),
                new CallRetSourceSpec(getLongShared, null),
                new CallRetSourceSpec(getStringSetShared, null),
                new CallRetSourceSpec(queryDatabase, null),
                new CallRetSourceSpec(queryWithFactoryDatabase, null),
                new CallRetSourceSpec(rawQueryDatabase, null),
                new CallRetSourceSpec(rawQueryWithFactoryDatabase, null) };

        public SinkSpec[] getSinkSpecs() {
```

```
		return sinkSpecs;
	}

	public SinkSpec[] getTransientSinkSpecs() {
		return transientSinkSpecs;
	}

	private static MethodNamePattern[] callBacks = new MethodNamePattern[] {};

	public static void addPossibleListeners(ClassHierarchy cha) {
		Set<String> ignoreMethods = new HashSet<String>();
		ignoreMethods.add("<init>");
		ignoreMethods.add("<clinit>");
		ignoreMethods.add("registerNatives");
		ignoreMethods.add("getClass");
		ignoreMethods.add("hashCode");
		ignoreMethods.add("equals");
		ignoreMethods.add("clone");
		ignoreMethods.add("toString");
		ignoreMethods.add("notify");
		ignoreMethods.add("notifyAll");
		ignoreMethods.add("finalize");
		ignoreMethods.add("wait");

		List<MethodNamePattern> moreEntryPointSpecs = new
ArrayList<MethodNamePattern>();

		// add default entrypoints from AndroidSpecs.entrypointSpecs
		// Currently adds methods even if they exist in the ignnoreMethods
		// set.
		for (MethodNamePattern mnp : defaultCallbacks) {
			moreEntryPointSpecs.add(mnp);
		}

		for (IClass ic : cha) {
			if (!LoaderUtils.fromLoader(ic,
ClassLoaderReference.Application)) {
				continue;
			}

			// finds all *Listener classes and fetches all methods for the
			// listener
			if (ic.getName().getClassName().toString().endsWith("Listener"))
{
				for (IMethod im : ic.getAllMethods()) {
					// TODO: add isAbstract()?
					if (!ignoreMethods.contains(im.getName().toString())
							&& !im.isPrivate()) {
						moreEntryPointSpecs
		.add(new MethodNamePattern(ic.getName().toString(),
im.getName().toString()));
					}
				}
			}
```

```
                    // not a listener, just find all the methods that start with
                    // "on____"
                    else {
                          for (IMethod im : ic.getAllMethods()) {
                                // TODO: add isAbstract()?
                                if (!ignoreMethods.contains(im.getName().toString())
                                            &&
im.getName().toString().startsWith("on")
                                            && !im.isPrivate()) {
                                      moreEntryPointSpecs
                                                  .add(new
MethodNamePattern(ic.getName()
                                                              .toString(),
im.getName().toString()));
                                }
                          }
                    }
             }

             // entrypointSpecs =
             callBacks = moreEntryPointSpecs
                          .toArray(new
MethodNamePattern[moreEntryPointSpecs.size()]);

      }

      public static MethodNamePattern[] getCallBacks() {
            return callBacks;
      }

}
```

LeakageAnalysis.java

```
package org.distibuteme;

import java.io.IOException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

import org.distibuteme.flow.FlowAnalysis;
import org.distibuteme.flow.InflowAnalysis;
import org.distibuteme.flow.OutflowAnalysis;
import org.distibuteme.util.CGAnalysisContext;
import org.scandroid.domain.CodeElement;
import org.scandroid.domain.DomainElement;
import org.scandroid.domain.IFDSTaintDomain;
import org.scandroid.flow.types.FlowType;
import org.scandroid.spec.ISpecs;
import org.scandroid.util.AndroidAnalysisContext;
import org.scandroid.util.CLISCanDroidOptions;
```

```
import org.scandroid.util.EntryPoints;
import org.scandroid.util.IEntryPointSpecifier;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.google.common.collect.Lists;
import com.ibm.wala.dataflow.IFDS.TabulationResult;
import com.ibm.wala.ipa.callgraph.CGNode;
import com.ibm.wala.ipa.callgraph.Entrypoint;
import com.ibm.wala.ipa.callgraph.propagation.InstanceKey;
import com.ibm.wala.ipa.cfg.BasicBlockInContext;
import com.ibm.wala.ssa.analysis.IExplodedBasicBlock;
import com.ibm.wala.util.MonitorUtil.IProgressMonitor;

public class LeakageAnalysis {
	private static final Logger logger = LoggerFactory
			.getLogger(LeakageAnalysis.class);

	public static void main(String[] args) throws Exception {
		CLISCanDroidOptions options = new CLISCanDroidOptions(args, true);
		logger.info("Loading app.");

		AndroidAnalysisContext analysisContext = new AndroidAnalysisContext(
				options);

		final List<Entrypoint> entrypoints = EntryPoints
				.defaultEntryPoints(analysisContext.getClassHierarchy());

		for (Entrypoint entry : entrypoints) {
			logger.info("Entry point: " + entry);
		}

		if (options.separateEntries()) {
			int i = 1;
			for (final Entrypoint entry : entrypoints) {
				CGAnalysisContext<IExplodedBasicBlock> cgContext = new
CGAnalysisContext<IExplodedBasicBlock>(
						analysisContext, new IEntryPointSpecifier() {
							@Override
							public List<Entrypoint> specify(
									AndroidAnalysisContext
analysisContext) {
								return
Lists.newArrayList(entry);
							}
						});
				logger.info("** Processing entry point " + i + "/"
						+ entrypoints.size() + ": " + entry);
				Map<InstanceKey, String> map = TransientAnalysis
						.runAnalysis(cgContext);
				logger.info("map " + map.size());
				for (Entry<InstanceKey, String> key : map.entrySet()) {
					logger.info("map key= " + key.getKey() + " "
							+ key.getValue());
```

```
                        }

                        analyze(cgContext, null);
                        i++;
                    }
                } else {
                    CGAnalysisContext<IExplodedBasicBlock> cgContext = new
CGAnalysisContext<IExplodedBasicBlock>(
                            analysisContext, new IEntryPointSpecifier() {
                                @Override
                                public List<Entrypoint> specify(
                                        AndroidAnalysisContext
analysisContext) {
                                    return entrypoints;
                                }
                            });
                    Map<InstanceKey, String> map = TransientAnalysis
                            .runAnalysis(cgContext);
                    logger.info("map " + map.size());
                    for (Entry<InstanceKey, String> key : map.entrySet()) {
                        logger.info("map key= " + key.getKey() + " "
                                + key.getValue());
                    }
                    analyze(cgContext, null);
                }
        }

        public static int analyze(
                    CGAnalysisContext<IExplodedBasicBlock> analysisContext,
                    IProgressMonitor monitor) throws IOException {
                try {
                    logger.info("Supergraph size = "
                            + analysisContext.graph.getNumberOfNodes());

                    Map<InstanceKey, String> prefixes;
                    if (analysisContext.getOptions().stringPrefixAnalysis()) {
                        logger.info("Running prefix analysis.");
                        prefixes = TransientAnalysis.runAnalysisHelper(
                                analysisContext.cg, analysisContext.pa);
                        logger.info("Number of prefixes = " +
prefixes.values().size());
                    } else {
                        prefixes = new HashMap<InstanceKey, String>();
                    }

                    ISpecs specs = new AndroidSpecs();

                    logger.info("Running inflow analysis.");
                    Map<BasicBlockInContext<IExplodedBasicBlock>,
Map<FlowType<IExplodedBasicBlock>, Set<CodeElement>>> initialTaints = InflowAnalysis
                            .analyze(analysisContext, prefixes, specs);

                    logger.info("  Initial taint size = " + initialTaints.size());
```

```
                    Logger.info("Running flow analysis.");
                    IFDSTaintDomain<IExplodedBasicBlock> domain = new
IFDSTaintDomain<IExplodedBasicBlock>();
                    TabulationResult<BasicBlockInContext<IExplodedBasicBlock>,
CGNode, DomainElement> flowResult = FlowAnalysis
                                .analyze(analysisContext, initialTaints, domain,
monitor);

                    Logger.info("Running outflow analysis.");
                    Map<FlowType<IExplodedBasicBlock>,
Set<FlowType<IExplodedBasicBlock>>> permissionOutflow = new OutflowAnalysis(
                                analysisContext, specs).analyze(flowResult, domain);
                    Logger.info("  Permission outflow size = "
                                + permissionOutflow.size());

                    Logger.info("");

      Logger.info("================================================================"
);
                    Logger.info("");

                    for (Map.Entry<BasicBlockInContext<IExplodedBasicBlock>,
Map<FlowType<IExplodedBasicBlock>, Set<CodeElement>>> e : initialTaints
                                .entrySet()) {
                          Logger.info(e.getKey().toString());
                          for (Map.Entry<FlowType<IExplodedBasicBlock>,
Set<CodeElement>> e2 : e
                                      .getValue().entrySet()) {
                                Logger.info(e2.getKey() + " <- " + e2.getValue());
                          }
                    }
                    for (Map.Entry<FlowType<IExplodedBasicBlock>,
Set<FlowType<IExplodedBasicBlock>>> e : permissionOutflow
                                .entrySet()) {
                          Logger.info(e.getKey().toString());
                          for (FlowType t : e.getValue()) {
                                Logger.info("    --> " + t);
                          }
                    }

                    return permissionOutflow.size();
              } catch (com.ibm.wala.util.debug.UnimplementedError e) {
                    Logger.error("exception during analysis", e);
              }
              return 0;
      }
}
```

TransientAnalysis.java

```
package org.distibuteme;

import java.util.ArrayList;
import java.util.HashMap;
```

```
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

import org.scandroid.prefixtransfer.InstanceKeySite;
import org.scandroid.prefixtransfer.PrefixTransferFunctionProvider;
import org.scandroid.prefixtransfer.PrefixVariable;
import org.scandroid.prefixtransfer.TransientTransferGraph;
import org.scandroid.util.CGAnalysisContext;
import org.scandroid.util.EmptyProgressMonitor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


import com.ibm.wala.dataflow.graph.DataflowSolver;
import com.ibm.wala.dataflow.graph.IKilldallFramework;
import com.ibm.wala.dataflow.graph.ITransferFunctionProvider;
import com.ibm.wala.ipa.callgraph.CallGraph;
import com.ibm.wala.ipa.callgraph.propagation.InstanceKey;
import com.ibm.wala.ipa.callgraph.propagation.PointerAnalysis;
import com.ibm.wala.ssa.analysis.IExplodedBasicBlock;
import com.ibm.wala.util.CancelException;
import com.ibm.wala.util.CancelRuntimeException;
import com.ibm.wala.util.graph.Graph;


public class TransientAnalysis {
      private static final Logger logger =
LoggerFactory.getLogger(TransientAnalysis.class);

    public static Map<InstanceKey,String>
runAnalysis(CGAnalysisContext<IExplodedBasicBlock> analysisContext) throws
CancelRuntimeException
    {
        return runAnalysisHelper(analysisContext.cg, analysisContext.pa);
    }

    public static ArrayList<InstanceKey> locateKeys(Map<InstanceKey,String> prefixes,
String s) {
        ArrayList<InstanceKey> keylist = new ArrayList<InstanceKey>();
        for (Entry<InstanceKey,String> e : prefixes.entrySet()) {
            if (e.getValue().contains(s))
                keylist.add(e.getKey());
        }
        return keylist;
    }

    public static Map<InstanceKey,String> runAnalysisHelper(CallGraph cg,
PointerAnalysis pa) throws CancelRuntimeException
    {

        logger.info("*******************************************************");
        logger.info("* Transient Analysis*");
```

```
        final Graph<InstanceKeySite> g = new TransientTransferGraph(pa);
        Logger.info("* The Graph:                                                *");
        Logger.info("************************************************************");
        Iterator<InstanceKeySite> iksI = g.iterator();

        final PrefixTransferFunctionProvider tfp = new
PrefixTransferFunctionProvider();

        IKilldallFramework<InstanceKeySite, PrefixVariable> framework = new
IKilldallFramework<InstanceKeySite, PrefixVariable>()
        {

            public Graph<InstanceKeySite> getFlowGraph() {
                return g;
            }

            public ITransferFunctionProvider<InstanceKeySite, PrefixVariable>
getTransferFunctionProvider() {
                return tfp;
            }

        };

        DataflowSolver<InstanceKeySite, PrefixVariable> dfs = new
DataflowSolver<InstanceKeySite, PrefixVariable>(framework){

            @Override
            protected PrefixVariable makeEdgeVariable(InstanceKeySite src,
                    InstanceKeySite dst) {
                return new PrefixVariable();
            }

            @Override
            protected PrefixVariable makeNodeVariable(InstanceKeySite n,
                    boolean IN) {
                PrefixVariable var = new PrefixVariable();
                return var;
            }

            @Override
            protected PrefixVariable[] makeStmtRHS(int size) {
                return new PrefixVariable[size];
            }

        };

        Logger.info("\n************************************************");
        Logger.info("* Running Analysis");

        try {
            dfs.solve(new EmptyProgressMonitor());
        } catch (CancelException e) {
            throw new CancelRuntimeException(e);
```

```
        }
        Map<InstanceKey,String> keys = new HashMap<InstanceKey,String>();
        iksI = g.iterator();
        while (iksI.hasNext()) {
            InstanceKeySite iks = iksI.next();
            keys.put((InstanceKey)
pa.getInstanceKeyMapping().getMappedObject(iks.instanceID()),
dfs.getOut(iks).knownPrefixes.get(iks.instanceID()));
        }

        return keys;
    }


}
```

TransientContextSelector.java

```
package org.distibuteme.flow;

import com.ibm.wala.classLoader.CallSiteReference;
import com.ibm.wala.classLoader.IMethod;
import com.ibm.wala.ipa.callgraph.AnalysisOptions;
import com.ibm.wala.ipa.callgraph.CGNode;
import com.ibm.wala.ipa.callgraph.Context;
import com.ibm.wala.ipa.callgraph.impl.DefaultContextSelector;
import com.ibm.wala.ipa.callgraph.propagation.InstanceKey;
import com.ibm.wala.ipa.callgraph.propagation.NormalAllocationInNode;
import com.ibm.wala.ipa.callgraph.propagation.ReceiverInstanceContext;
import com.ibm.wala.ipa.callgraph.propagation.cfa.CallerSiteContext;
import com.ibm.wala.ipa.callgraph.propagation.cfa.CallerSiteContextPair;
import com.ibm.wala.ipa.cha.IClassHierarchy;
import com.ibm.wala.types.ClassLoaderReference;
import com.ibm.wala.util.intset.IntSet;

public class TransientContextSelector extends DefaultContextSelector {

    public TransientContextSelector(AnalysisOptions options, IClassHierarchy cha) {
            super(options, cha);
      }

    @Override
    public Context getCalleeTarget(CGNode caller, CallSiteReference site,
            IMethod callee, InstanceKey[] receivers) {

if(callee.getSignature().equals("java.lang.StringBuilder.toString()Ljava/lang/String;
") ||

callee.getSignature().equals("java.lang.StringBuilder.append(Ljava/lang/String;)Ljava
/lang/StringBuilder;") ||

callee.getSignature().equals("java.lang.String.valueOf(Ljava/lang/Object;)Ljava/lang/
String;") ||
```

```
callee.getSignature().equals("java.lang.String.toString()Ljava/lang/String;") ||

callee.getSignature().equals("android.content.SharedPreferences.getString(Ljava/lang/
String;Ljava/lang/String;)Ljava/lang/String;") ||

callee.getSignature().equals("android.content.SharedPreferences.Editor.putString(Ljav
a/lang/String;Ljava/lang/String;)Ljava/lang/String;"))
        {

            if(receivers[0] instanceof NormalAllocationInNode)
            {


if(((NormalAllocationInNode)receivers[0]).getSite().getDeclaredType().getClassLoader(
).equals(ClassLoaderReference.Application)&&!((NormalAllocationInNode)receivers[0]).g
etNode().getMethod().getSignature().contains("android.support")){
                    // create a context based on the site and the receiver
                    return new CallerSiteContextPair(caller,site,new
ReceiverInstanceContext(receivers[0]));
                }
            }
            else
if(callee.getSignature().equals("java.lang.String.valueOf(Ljava/lang/Object;)Ljava/la
ng/String;") ||

callee.getSignature().equals("java.lang.String.toString()Ljava/lang/String;"))
            {
                return new CallerSiteContext(caller,site);
            }
        }
        return super.getCalleeTarget(caller, site, callee, receivers);
    }

    @Override
    public IntSet getRelevantParameters(CGNode node, CallSiteReference call) {
       return super.getRelevantParameters(node,call);
    }
}
```

CGAnalysisContext.java

```
package org.distibuteme.util;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Deque;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
```

```
import org.distibuteme.flow.TransientContextSelector;
import org.scandroid.domain.CodeElement;
import org.scandroid.domain.FieldElement;
import org.scandroid.domain.InstanceKeyElement;
import org.scandroid.util.AndroidAnalysisContext;
import org.scandroid.util.IEntryPointSpecifier;
import org.scandroid.util.ISCanDroidOptions;
import org.scandroid.util.LoaderUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.google.common.collect.Queues;
import com.ibm.wala.classLoader.IClass;
import com.ibm.wala.classLoader.IField;
import com.ibm.wala.classLoader.IMethod;
import com.ibm.wala.dalvik.classLoader.DexIRFactory;
import com.ibm.wala.dataflow.IFDS.ICFGSupergraph;
import com.ibm.wala.dataflow.IFDS.ISupergraph;
import com.ibm.wala.ipa.callgraph.AnalysisCache;
import com.ibm.wala.ipa.callgraph.AnalysisOptions;
import com.ibm.wala.ipa.callgraph.AnalysisScope;
import com.ibm.wala.ipa.callgraph.CGNode;
import com.ibm.wala.ipa.callgraph.CallGraph;
import com.ibm.wala.ipa.callgraph.Entrypoint;
import com.ibm.wala.ipa.callgraph.impl.Everywhere;
import com.ibm.wala.ipa.callgraph.impl.PartialCallGraph;
import com.ibm.wala.ipa.callgraph.propagation.ConcreteTypeKey;
import com.ibm.wala.ipa.callgraph.propagation.InstanceKey;
import com.ibm.wala.ipa.callgraph.propagation.PointerKey;
import com.ibm.wala.ipa.callgraph.propagation.SSAPropagationCallGraphBuilder;
import com.ibm.wala.ipa.cha.ClassHierarchy;
import com.ibm.wala.ssa.IRFactory;
import com.ibm.wala.ssa.ISSABasicBlock;
import com.ibm.wala.ssa.SSACFG;
import com.ibm.wala.ssa.SSACFG.BasicBlock;
import com.ibm.wala.ssa.SSAInstruction;
import com.ibm.wala.types.ClassLoaderReference;
import com.ibm.wala.types.TypeReference;
import com.ibm.wala.util.Predicate;
import com.ibm.wala.util.collections.HashSetFactory;
import com.ibm.wala.util.graph.GraphSlicer;
import com.ibm.wala.util.intset.OrdinalSet;
import com.ibm.wala.util.warnings.Warning;
import com.ibm.wala.util.warnings.Warnings;

public class CGAnalysisContext<E extends ISSABasicBlock> extends
org.scandroid.util.CGAnalysisContext<E>{
      private static final Logger logger =
LoggerFactory.getLogger(CGAnalysisContext.class);

      public CGAnalysisContext(AndroidAnalysisContext analysisContext,
IEntryPointSpecifier specifier)
                  throws IOException {
```

```
		this(analysisContext, specifier, new ArrayList<InputStream>());
	}

	public CGAnalysisContext(AndroidAnalysisContext analysisContext,
IEntryPointSpecifier specifier,
			Collection<InputStream> extraSummaries) throws IOException {
		super(analysisContext, specifier, extraSummaries);
		final AnalysisScope scope = analysisContext.getScope();
		final ClassHierarchy cha = analysisContext.getClassHierarchy();
		final ISCanDroidOptions options = analysisContext.getOptions();

		entrypoints = specifier.specify(analysisContext);
		AnalysisOptions analysisOptions = new AnalysisOptions(scope,
entrypoints);
		for (Entrypoint e : entrypoints) {
			logger.debug("Entrypoint: " + e);
		}
		analysisOptions.setReflectionOptions(options.getReflectionOptions());

		AnalysisCache cache = new AnalysisCache((IRFactory<IMethod>) new
DexIRFactory());

		SSAPropagationCallGraphBuilder cgb;

		if (null != options.getSummariesURI()) {
			extraSummaries.add(new FileInputStream(new
File(options.getSummariesURI())));
		}

		cgb =
AndroidAnalysisContext.makeVanillaZeroOneCFABuilder(analysisOptions, cache,    cha,
scope,
				new TransientContextSelector(analysisOptions, cha), null,
null, null);

		if (analysisContext.getOptions().cgBuilderWarnings()) {
			// CallGraphBuilder construction warnings
			for (Iterator<Warning> wi = Warnings.iterator(); wi.hasNext();) {
				Warning w = wi.next();
				logger.warn(w.getMsg());
			}
		}
		Warnings.clear();

		logger.info("*************************");
		logger.info("* Building Call Graph   *");
		logger.info("*************************");

		boolean graphBuilt = true;
		try {
			cg = cgb.makeCallGraph(cgb.getOptions());
		} catch (Exception e) {
			graphBuilt = false;
			if (!options.testCGBuilder()) {
```

```
                        throw new RuntimeException(e);
                    } else {
                        e.printStackTrace();
                    }
                }

                if (options.testCGBuilder()) {
                    int status = graphBuilt ? 0 : 1;
                    System.exit(status);
                }

                pa = cgb.getPointerAnalysis();
                partialGraph = GraphSlicer.prune(cg, new Predicate<CGNode>() {
                    @Override
                    // CallGraph composed of APK nodes
                    public boolean test(CGNode node) {
                        return LoaderUtils.fromLoader(node,
ClassLoaderReference.Application) || node.getMethod().isSynthetic();
                    }
                });
                if (options.includeLibrary()) {
                    graph = (ISupergraph) ICFGSupergraph.make(cg, cache);
                } else {

                    Collection<CGNode> nodes = HashSetFactory.make();
                    for (Iterator<CGNode> nIter = partialGraph.iterator();
nIter.hasNext();) {
                        nodes.add(nIter.next());
                    }
                    CallGraph pcg = PartialCallGraph.make(cg,
cg.getEntrypointNodes(), nodes);
                    graph = (ISupergraph) ICFGSupergraph.make(pcg, cache);
                }

                oneLevelGraph = GraphSlicer.prune(cg, new Predicate<CGNode>() {
                    @Override
                    public boolean test(CGNode node) {
                        // Node in APK
                        if (LoaderUtils.fromLoader(node,
ClassLoaderReference.Application)) {
                            return true;
                        } else {
                            Iterator<CGNode> n = cg.getPredNodes(node);
                            while (n.hasNext()) {
                                // Primordial node has a successor in APK
                                if (LoaderUtils.fromLoader(n.next(),
ClassLoaderReference.Application))
                                    return true;
                            }
                            n = cg.getSuccNodes(node);
                            while (n.hasNext()) {
                                // Primordial node has a predecessor in APK
                                if (LoaderUtils.fromLoader(n.next(),
ClassLoaderReference.Application))
```

```
                                        return true;
                                }
                                return false;
                        }
                }
        });

        systemToApkGraph = GraphSlicer.prune(cg, new Predicate<CGNode>() {
                @Override
                public boolean test(CGNode node) {

                        if (LoaderUtils.fromLoader(node,
ClassLoaderReference.Primordial)) {
                                Iterator<CGNode> succs = cg.getSuccNodes(node);
                                while (succs.hasNext()) {
                                        CGNode n = succs.next();

                                        if (LoaderUtils.fromLoader(n,
ClassLoaderReference.Application)) {
                                                return true;
                                        }
                                }
                                // Primordial method, with no link to APK code:
                                return false;
                        } else if (LoaderUtils.fromLoader(node,
ClassLoaderReference.Application)) {
                                // see if this is an APK method that was
                                // invoked by a Primordial method:
                                Iterator<CGNode> preds = cg.getPredNodes(node);
                                while (preds.hasNext()) {
                                        CGNode n = preds.next();

                                        if (LoaderUtils.fromLoader(n,
ClassLoaderReference.Primordial)) {
                                                return true;
                                        }
                                }
                                // APK code, no link to Primordial:
                                return false;
                        }

                        // who knows, not interesting:
                        return false;
                }
        });

        if (options.stdoutCG()) {
                for (Iterator<CGNode> nodeI = cg.iterator(); nodeI.hasNext();) {
                        CGNode node = nodeI.next();

                        logger.debug("CGNode: " + node);
                        for (Iterator<CGNode> succI = cg.getSuccNodes(node);
succI.hasNext();) {
```

```
                                Logger.debug("\tSuccCGNode: " +
succI.next().getMethod().getSignature());
                            }
                        }
                }
                for (Iterator<CGNode> nodeI = cg.iterator(); nodeI.hasNext();) {
                        CGNode node = nodeI.next();
                        if (node.getMethod().isSynthetic()) {
                                Logger.trace("Synthetic Method: {}",
node.getMethod().getSignature());
                                Logger.trace("{}",
node.getIR().getControlFlowGraph().toString());
                                SSACFG ssaCFG = node.getIR().getControlFlowGraph();
                                int totalBlocks = ssaCFG.getNumberOfNodes();
                                for (int i = 0; i < totalBlocks; i++) {
                                        Logger.trace("BLOCK #{}", i);
                                        BasicBlock bb = ssaCFG.getBasicBlock(i);

                                        for (SSAInstruction ssaI : bb.getAllInstructions())
{
                                                Logger.trace("\tInstruction: {}", ssaI);
                                        }
                                }
                        }
                }
        }

        public Set<CodeElement> codeElementsForInstanceKey(InstanceKey rootIK) {
                Set<CodeElement> elts = HashSetFactory.make();
                Deque<InstanceKey> iks = Queues.newArrayDeque();
                iks.push(rootIK);

                while (!iks.isEmpty()) {
                        InstanceKey ik = iks.pop();
                        Logger.debug("getting code elements for {}", ik);
                        elts.add(new InstanceKeyElement(ik));
                        final IClass clazz = ik.getConcreteType();
                        final TypeReference typeRef = clazz.getReference();
                        // If an array, recur down into the structure
                        if (typeRef.isArrayType()) {
                                if (typeRef.getArrayElementType().isPrimitiveType()) {
                                        // don't do anything for primitive contents
                                        continue;
                                }
                                OrdinalSet<InstanceKey> pointsToSet =

        pa.getPointsToSet(pa.getHeapModel().getPointerKeyForArrayContents(ik));
                                if (pointsToSet.isEmpty()) {
                                        Logger.debug("pointsToSet empty for array contents,
creating InstanceKey manually");
                                        final IClass contentsClass =
pa.getClassHierarchy().lookupClass(typeRef.getArrayElementType());
                                        if (contentsClass.isInterface()) {
```

```
                                        for (IClass implementor :
analysisContext.concreteClassesForInterface(contentsClass)) {
                                                final InstanceKey contentsIK = new
ConcreteTypeKey(implementor);
                                                final InstanceKeyElement elt = new
InstanceKeyElement(contentsIK);
                                                if (!elts.contains(elt)) {
                                                      elts.add(elt);
                                                      iks.push(contentsIK);
                                                }
                                        }
                                  } else {
                                        InstanceKey contentsIK = new
ConcreteTypeKey(contentsClass);
                                        final InstanceKeyElement elt = new
InstanceKeyElement(contentsIK);
                                        if (!elts.contains(elt)) {
                                              elts.add(elt);
                                              iks.push(contentsIK);
                                        }
                                  }
                            } else {
                                  for (InstanceKey contentsIK : pointsToSet) {
                                        final InstanceKeyElement elt = new
InstanceKeyElement(contentsIK);
                                        if (!elts.contains(elt)) {
                                              elts.add(elt);
                                              iks.push(contentsIK);
                                        }
                                  }
                            }
                            continue;
                      }
                      for (IField field : clazz.getAllInstanceFields()) {
                            Logger.debug("adding elements for field {}", field);
                            final TypeReference fieldTypeRef =
field.getFieldTypeReference();
                            elts.add(new FieldElement(ik, field.getReference()));
                            final IClass fieldClass =
analysisContext.getClassHierarchy().lookupClass(fieldTypeRef);
                            if (fieldTypeRef.isPrimitiveType() || fieldClass == null)
{
                                  continue;
                            } else if (fieldTypeRef.isArrayType()) {
                                  PointerKey pk =
pa.getHeapModel().getPointerKeyForInstanceField(ik, field);
                                  final OrdinalSet<InstanceKey> pointsToSet =
pa.getPointsToSet(pk);
                                  if (pointsToSet.isEmpty()) {
                                        Logger.debug("pointsToSet empty for array
field, creating InstanceKey manually");
                                        InstanceKey fieldIK = new
ConcreteTypeKey(pa.getClassHierarchy().lookupClass(fieldTypeRef));
```

```
                                        final InstanceKeyElement elt = new
InstanceKeyElement(fieldIK);
                                        if (!elts.contains(elt)) {
                                                elts.add(elt);
                                                iks.push(fieldIK);
                                        }
                                } else {
                                        for (InstanceKey fieldIK : pointsToSet) {
                                                final InstanceKeyElement elt = new
InstanceKeyElement(fieldIK);
                                                if (!elts.contains(elt)) {
                                                        elts.add(elt);
                                                        iks.push(fieldIK);
                                                }
                                        }
                                }
                        } else if (fieldTypeRef.isReferenceType()) {
                                PointerKey pk =
pa.getHeapModel().getPointerKeyForInstanceField(ik, field);
                                final OrdinalSet<InstanceKey> pointsToSet =
pa.getPointsToSet(pk);
                                if (pointsToSet.isEmpty() &&
!analysisContext.getClassHierarchy().isInterface(fieldTypeRef)) {
                                        logger.debug("pointsToSet empty for reference
field, creating InstanceKey manually");
                                        InstanceKey fieldIK = new
ConcreteTypeKey(fieldClass);
                                        final InstanceKeyElement elt = new
InstanceKeyElement(fieldIK);
                                        if (!elts.contains(elt)) {
                                                elts.add(elt);
                                                iks.push(fieldIK);
                                        }
                                } else {
                                        for (InstanceKey fieldIK : pointsToSet) {
                                                final InstanceKeyElement elt = new
InstanceKeyElement(fieldIK);
                                                if (!elts.contains(elt)) {
                                                        elts.add(elt);
                                                        iks.push(fieldIK);
                                                }
                                        }
                                }
                        }
                }
        }
        return elts;
    }

    public ISCanDroidOptions getOptions() {
        return analysisContext.getOptions();
    }

    public ClassHierarchy getClassHierarchy() {
```

```
            return analysisContext.getClassHierarchy();
        }

        public AnalysisScope getScope() {
            return analysisContext.getScope();
        }

        public List<Entrypoint> getEntrypoints() {
            return entrypoints;
        }

        public CGNode nodeForMethod(IMethod method) {
            return cg.getNode(method, Everywhere.EVERYWHERE);
        }
}
```

العنوان:	Detection Of Redestributed Malware Behavior In Android App Versions
المؤلف الرئيسي:	Al Salehi, Alaa
مؤلفين آخرين:	Abu Samra, Aiman Ahmed(Advisor)
التاريخ الميلادي:	2015
موقع:	غزة
الصفحات:	1 - 89
رقم MD:	768547
نوع المحتوى:	رسائل جامعية
اللغة:	English
الدرجة العلمية:	رسالة ماجستير
الجامعة:	الجامعة الإسلامية (غزة)
الكلية:	كلية الهندسة
الدولة:	فلسطين
قواعد المعلومات:	Dissertations
مواضيع:	هندسة الحاسوب، البرمجيات الخبيثة، برنامج أندرويد
رابط:	https://search.mandumah.com/Record/768547

اكتشاف السلوك الخبيث الموزع على عدة إصدارات من برنامج أندرويد

DETECTION OF REDESTRIBUTED MALWARE BEHAVIOR IN ANDROID APP VERSIONS

By

Alaa Al Salehi

120090707

A Master of Science Thesis Proposal

Supervisor:

Dr. Aiman Abu Samra

Computer Engineering Department

Islamic University

Gaza

Palestine

Feb, 2015

اكتشاف السلوك الخبيث الموزع على عدة إصدارات من برنامج أندرويد

DETECTION OF REDESTRIBUTED MALWARE BEHAVIOR IN ANDROID APP VERSIONS

By

Alaa Al Salehi

120090707

A Master of Science Thesis Proposal

Supervisor:

Dr. Aiman Abu Samra

Computer Engineering Department

Islamic University

Gaza

Palestine

Feb, 2015

الملخص

مع زيادة أهمية البيانات الموجودة على الأجهزة الذكية زادت البرمجيات الخبيثة التي تحاول الحصول على هذه المعلومات لاستغلالها إما عن طريق تحليل البيانات وتعقب أصحاب الأجهزة لأغراض إعلانية أو أغراض تسويقية أو عن طريق ابتزاز المستخدمين. وزيادة هذه البرمجيات أدت إلى زيادة في أهمية الأبحاث التي تعمل على تحليل البرمجيات الخبيثة واكتشاف هذا النوع من السلوك.

هذه الأطروحة تدرس تحوير السلوك الخبيث الذي يتم فيه توزيع مصدر البيانات ونقطة فقد البيانات على الإصدارات المختلفة من البرنامج باستخدام وسائل التخزين المحلية في البرامج لتخزين جزء أو كل من هذه البيانات الحيوية.

قامت هذه الأطروحة ببناء خوارزمية سميت Distributed Malware Detection Algorithm واختصاراً DMDA هذه الخوارزمية تقوم بتحليل البيانات وتحديد مصادر البيانات ونقاط الفقد الانتقالية التي يتم استخدامها للتعتيم على عملية كشف مناطق فقد البيانات على مستوى أكثر من نسخة من البرنامج.

تمت تجربة هذه الخوارزمية على عينة من تطبيقات الأندرويد المنشورة على سوق Google Play تحتوي على 100 تطبيق كل تطبيق تم أخذ نسختين منه وتم كشف عما يزيد 150 مصدر انتقالي للبيانات و200 نقطة فقد للبيانات انتقالي ونقطة فقد وحيدة وتم تجريب البرامج نفسها بإصداراتها على 56 مضاداً للأكواد الخبيثة ولم تجد أياً منها أي مشكلة في أي من إصداراتها.

Abstract

The importance of data stored on smart devices can make malware apps that are trying to get this information to be exploited in either the data analysis and tracking devices for the purposes of the owners of advertising or marketing purposes or for blackmailing purposes of users. Increasing malwares has led to an increase in the importance of research work on malware analysis and the discovery of this kind of behavior.

This thesis is considered altered attack method, which distribute of the data source and the point of loss of data on different versions of the app using local storage to storing part or all of vital data to leak in future.

This thesis will introduce Distributed Malware Detection Algorithm (DMDA), which is an algorithm to detect distributed malware on app versions and propose new way to analyze application against redistributed malware.

DMDA created to analyze the data and identify transitional losses points that are used to gloss over the algorithm sources.

We test this algorithm on a sample of Android applications published on the Google Play market containing 100 application; every application has two version of it. The algorithm was revealed 150 transient data source, 200 transient loss of data point and 2 leakage of data. This dataset was checked by 56 anti-malware and none of them find any malicious code.

Dedication

To my parents, my family, my wife and to my baby Mohamed

Acknowledgement

I would like to acknowledge my thesis supervisors Dr. Aiman Abu Samra for his guidance and valuable help. I also want to acknowledge my college Mahmoud Al-kurdi who is help me with valuable resources.

Contents

Figures

1- Introduction

1.1 Topic Area

Smart phones are becoming more integrated and important part of people's daily lives due to their highly powerful computational capabilities, such as email applications, online banking and online shopping…etc.

Malware, short for malicious software, is one of the major security threats in information systems. Malware includes viruses, worms, Trojan horses, spyware, dishonest adware, most root kits, and other malicious and unwanted software [1].

Android is an OS for smart phone owned by Google Inc, Google wants Android to become dominant in smart phone field, so they create their market to be an open market for developers with easy conditions for publishing new apps. In addition, Google opened Android for company solutions –companies can deploy their own modification on Android OS, Also Google allows Android`s users to install apps from other markets –there is a lot of android markets like Amazon store, SildeMe, Aptoide,…etc - and even form a website –unknown source-.This makes android a great environment for developers, marketers, users and companies.

This tremendous increase unfortunately, also makes android target for Malware applications and application`s thieves. Malware applications become the main threat field because of large custom and private data can be collected form user smart phones like Identifiers Disclosure - individually phone number, International

Mobile Equipment Identity number (lMEI)-, SMS, call log , contacts, browser history, location and emails. In addition, Malware can misuse SMS for Premium messages and root exploits. [2, 3, 4, 5]

In addition to malware android is a hot business field for developers also repackage app can threat their businesses. There are several ways developers may lose potential revenue: a paid application may be “cracked” and released for free, a free application may be copied and re-released on other markets with changes to the ad libraries or even in the same market with changes on interface and services. That will cause ad revenue or paid price goes to the plagiarist.

1.2 Research Question

The popularity and adoption of Smart phones has greatly stimulated the spread of mobile malware, especially on the popular platforms such as Android. In light of their rapid growth, there is a pressing need to develop effective solutions. General countermeasures to Android malwares are currently limited to signature-based antivirus scanners, which efficiently detect known malwares, but they have serious shortcomings with repackaged, refectories and redistributed. These maybe on threads, on versions, on components or maybe on different applications.

So the question is how to detect these behaviors on apps?

1.3 Significance

Tremendous increase of android markets make it easy for anyone to publish apps and update these apps. There is also a rising danger associated with Malware applications at mobile devices, so the problem of detecting Malwares is an interesting topic. In fact, 86% of detected malwares are old malware repackaged in new apps [6]. However, the fact all antimalware and antivirus focus on the current app version and they do not count malwares distributed on different versions of the same application.

In this research, we introduce a way to detect distributed malware on app version application and propose new way to analyze application against redistributed malware.

1.4 Thesis Structure

This thesis is organized as follows:

Chapter 1; Introduction: In this chapter thesis provides an introduction about thesis problem, questions and significance, this chapter describes why we choose this title for thesis and the idea of proposed solution.

Chapter 2; Background and Related Work: This chapter provides a background about Android system, application and programming. It also talks about malwares in general and malware in Android applications, at the end of this chapter there is a group of related work in the same topic of this thesis.

Chapter 3; Research Approach and Tools: This chapter describes in theoretical view the most important used tools in this thesis; it provides readers with description about used applications.

Chapter 4; Attack model: This chapter describes the model assumed on attack and the idea of distribution malware behavior.

Chapter 5; Methodology Evaluation and Analysis. Here readers can show the used methodology for thesis, and how we prove the feasibility of our idea, details of DMDA algorithm, this chapter also provides details about experiments and it results, in addition, it provides more details about algorithm.

Chapter 6; Conclusion and Future Work: A complete conclusion has been written in this chapter; also, we talked about future work related to his topic.

References: this chapter is a list of all sources associated with thesis.

Appendices: In this chapter, author attaches sample on the attack thread and code implementation for the proposed algorithm.

2- Background and Related work

2.1 ANDROID BACKGROUND

Android is a modern mobile platform that is designed to be truly open platform. Android developers use advanced hardware and software, as well as local and remote data, exposed through the platform to bring innovation and value applications to consumers.

2.1.1 Android System Architecture

The architecture of Android is implemented as a software stack, customized for mobile devices. Figure 1 Android some of the most important components of this stack [7].

The core of the Android platform is a Linux kernel. The kernel is responsible for handling device drivers, resource access, memory process, power management and other typical OS duties. The kernel also acts as an abstraction layer between the hardware and other software stack.

On top of the kernel are several native C/C++ libraries and Dlaivk VM. On the top of this layer there is Application framework of android which is responsible of managing android component lifecycle and interaction between android applications and low level APIs like media framework, OpenGL and etc.. On top of application framework there is application layer which contains contact, phone, SMS and E-mail applications.

2.1.2 Android Application Entry points

Android provides a Software Developer Kit (SDK) to developers. This SDK exposes the API needed by developers to build applications. Unlike java application, that has one entry point for application –main method- and works on one program architecture, android application has multi-entry point and works on message passing architecture. These multi entry points are:

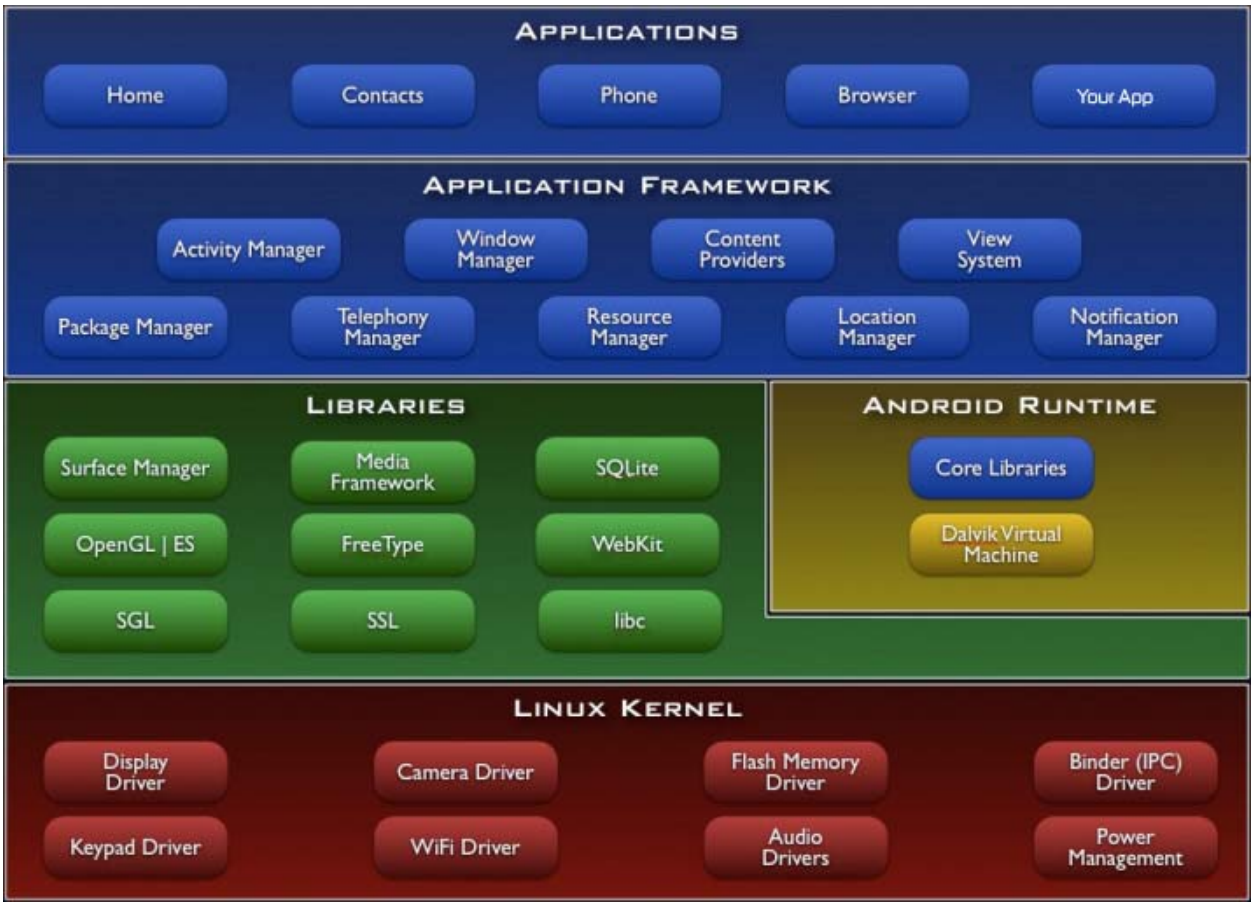

Figure 1 Android Architecture (Source Android developers) [8]

A. Activities

In Android, an activity is an essential components of an application. Every visual application should at least have one activity, the “main” activity even Launcher app-special application which is running when android user open their device- contains activity. Usually, an application has many activities and these can start each other, where each activity holds its state (e.g. start, stop or pause). An activity is the component that provides graphical user interface (GUI) for users.

Activity is one of the most complex and important component in android, android framework focuses on making activity optimized for users –by optimized battery and memory usage- and fixable for developers –by lifecycle callback- as much as possible. Figure 2: Activity Lifecycle explain how android framework manage activity lifecycle, onCreate and onDestroy called when activity start lives in memory and when activity ready to remove from memory. Onstart and OnStop called when activity start showed to user and hidden from user. OnResume and OnPause called when activity gain and lose focus of user. OnRestart called when activity started after stopped. Those are the main callback in activity lifecycle and there is others but less important [9].

B. Services

Services are components in Android that do not provide any user interface, and always run in the background to process long running operations. Services are

starting by other components –even in other applications- , so other component as activity or service can start a service. Android defines two types of Service bounded and unbounded. Bounded service life is independent on the component that started it. Unbounded Service does not depend on any component and any component can starting or stopping it.

C. Broadcast Receiver

Broadcast receiver is a mechanism that defines how Android operating system forwards its events to applications. The main usage of these broadcast receivers is inter-process communication and tracking of specific events (e.g. arrival of an SMS). Applications declare statically or dynamically their interest in receiving a certain event and accordingly the OS will try to deliver this event when it happens. Android defines two types of Broadcast Receivers: ordered and normal.

The normal Broadcast receiver is asynchronous and there is no order according it, which applications registered to get a broadcast, would receive the event first. As for the ordered ones, a priority can be set to require from the system to deliver the event to each app in a certain sequence, and some apps will get the event before others. This feature allows developers to capture and possibly modify the event`s carried data before it reaches to lower-priority consumers. In order case, an app can prevent other apps from getting specific event by aborting the

received data. Broadcast on android is one type of messaging on message passing architecture where sender send message to group of receivers.

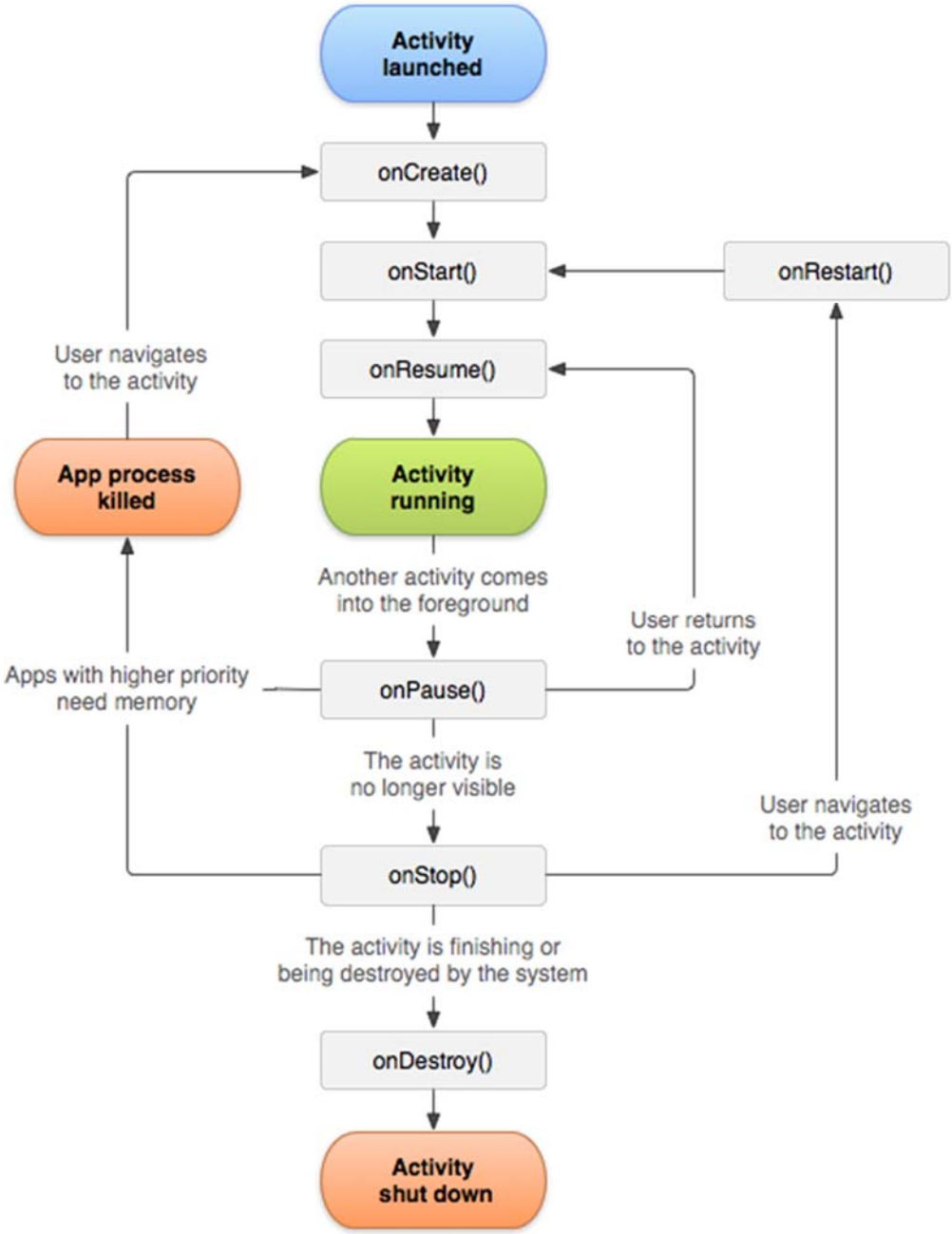

Figure 2: Activity Lifecycle (Source Android developers) [10]

2.1.3 Android Application Structure

Applications in android platform distributed in files called Android Packages (APKs). These files contain everything that the application needs to run from resources like images and XML files specifying UI layouts to the application code and metadata about what is the component of the application. APKs also include a manifest XML that specifies a number of metadata about the application, including its name, version information, the package (or namespace) of the code, the permissions it requires to execute, the component it contains and much more. Android applications are primarily developed in Java, sometimes native code may be used. The Java source code compiled to Java byte code and then converted into the Dalvik executable (DEX) format. Although similar to Java byte code, DEX byte code is incompatible with the Java virtual machine and instead runs on the Dalvik virtual machine. The conversion of Java byte code to DEX byte code is easily reversible and there are several tools can handle it.

2.1.4 Delvik VM

Android allows developers to run their application on top of virtual machine –known as Delvik VM-. Delvik VM created to handle limited memory size –about 60 MB only- this kind of VM can`t handle standard byte code files .class even compressed files .jar because of its limited size. It needs special pre-processing so it replace .jar .class with classes.dex and apk files. Those kind of files replace every string, every method name and every class name by id and lookup table. This

strategy reduce data loaded in memory and keep more rooms to the actual data in the application.

Android VM A.K.A Sandbox is a tool used in inter-application separation; every application runs in android must running alone on one VM. VM doing inter application division by two ways. First, every app has its different user ID. Second, every app is using its manifest file for to determine specific permissions.

VM Actually opens the gate of reverse engineers to reverse apks to classes.dex and resources. Again, reverse class.dex to classes, which mean inject malware behavior or ads in real and healthy app.

2.1.5 Android Storage Options

Android provides several options for you to save application data. The option you choose depends on your application needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires etc...

Android data storage options are the following: Shared Preferences, Internal Storage, External Storage, SQLite Databases and Network Connection [7]. The Shared Preferences provides a general framework that allows saving and retrieving persistent key-value pairs of primitive data types. Shared Preferences can used to save any primitive data: Booleans, floats, integers, longs, and strings. This data

will persist across user sessions. Internal Storage can used to save files directly on the device's internal storage. By default, files saved to the internal storage are private to your application and other applications cannot access them (nor can the user). When the user uninstalls your application, these files are removed. External Storage are world-readable and can be modified by the user. SQLite Databases used to save structural relation data and retrieve them using SQL standard with integrity constraint and indexing to fast retrieval when there is many data. In addition, android has versioning mechanism to upgrade and downgrade database, which help application to extend their data structure [11]. Network can used (when it is available) to store and retrieve data on your own web-based services.

Android provides a way to expose even your private data to other applications — with a content provider. A content provider is an optional component that exposes read/write access to application data for other applications, subject to whatever restrictions you want to impose.

2.2 Android Malwares

Malicious software is referred to as malware, classified by its nature as either computer virus, Trojan horse, worm, backdoor or rootkit. The most common malware types [12] are:

Virus: Code that that inserts itself into another program and replicates, that is, copies itself and infects other computers. Nowadays often used as a generic term that also includes worms and Trojans horses.

Worm: Self-replicating malware, which copies itself to other nodes in a network without user interaction using vulnerabilities. Worms do not attach themselves to an application like a virus do.

Trojan horse: Malicious program, which masquerades itself as being an application. Unlike viruses and worms, it does not replicate itself.

Rootkit: Software that enables continued privileged access to a computer while actively hiding its malicious activity from administrators by modifying the operating system functionality.

Backdoor: Specialized Trojan horse that masquerades itself as an installed program to enable remote access to a system and bypassing normal authentication. Additionally, backdoors attempts to remain undetected.

Spyware: Software that reveals private information about the user or computer system to eavesdroppers.

Bot: Piece of malware that allows the bot master, i.e. the author to remotely the infected system. Groups of infected systems that are controlled, which are denoted

as botnets, instructed by the bot master to perform various malicious activity such as distributed denial of services, stealing private information and sending spam.

2.3 Related work

Mobile security issues have gained much attention recently. Malware are available on both the official Android market and alternative ones [13]. Research efforts were made on detecting repackaged apps [14] or apps with known malicious behavior [15, 16]. Google also launched its malware filtering engine [17]. Information leakage is another major security threat for mobile devices. Kirin [18] detects apps whose permissions might indicate potential leakage.

In general, information leakage detection reveals the potential out bound propagation of sensitive information, which might be benign in many cases. Instead, component hijacking detection captures the information leakages resulted from an exploitation (i.e. sensitive data theft), in addition to other hijacking types.

Enck et al. introduced Ded [19] to convert Dalvik bytecode back to Java bytecode, and then used existing decompilers to obtain the source code of the apps for analysis.

Android mediates access to protect resources using a permission system. However, it's effectiveness hinges on app developers correctly implementing it. Chin et al. showed that apps might be exploitable when servicing external intents

[20]. They built ComDroid to identify publicly exported components and warn developers about the potential threats. For that, ComDroid checks app metadata and specific API usages.

As a result, warned public components are not necessarily exploitable or harmful (i.e. the openness can be by design or the component is not security critical). On the other hand, Android permission system is subject to several instances of the classic confused deputy attack [21]. As demonstrated by [22 ،23 ، 15], an unprivileged app can access permission-protected resources through privileged apps that do not check permissions. Grace et al. [15] employed an intra-procedural path-sensitive static analysis to discover permission leaks specific to stock apps from multiple device vendors.

Malware detection in general has two track static analysis and dynamic analysis. As Android applications are largely interactive and have a lot of interference between their components, dynamic detecting malware code would face scalability limitations as TaintDroid [24], where authors had to interact manually with each application. This eliminates techniques such as [25, 26, 27, 28] for detecting Android`s malwares applications. Therefore, we concentrate in this research on static analysis.

2.3.1 Static Analysis

Static analysis is an analysis of program application without executing the program. Static analysis of malware android application has three main categorization: Feature Based, Structure Based and Program Dependency Graph (PDG) Based.

2.3.1.1 Feature Based

Feature based approaches analyze a program and extract a set of features. Similarity between program and malware is detecting by comparing the extracted features from the programs. The features choice can vary significantly, from number or size of classes, methods, loops, or variables to included libraries.

Tesfay et Al. [29] Provided Anti-malware cloud that contains reputation for every version of every application using APK's hash code and depends on user Anti-malware reputation. Actually, there approach cannot handle repackaged APKs because simple change like space or comma in the APK content means completely different hash code.

This approach is limited -even with AI still need more investigation [30, 31]- and not realistic because it discards too much information about the structure of the programs.

2.3.1.2 Structure Based

Structure based systems convert programs into a stream of tokens and then compare the streams between two programs. By converting programs into a stream of tokens and ignoring easily, changed constructs such as comments, whitespace, and variable names, structure based systems detect plagiarism more robustly than feature-based systems.

Zhou et Al. [14] They work on DroidMOSS framework, it adopt a specialized hashing technique called fuzzy hashing. Instead of directly processing or comparing the entire (long) instruction sequences, it first condenses each sequence into one much shorter fingerprint. The similarity between two apps calculated based on the shorter fingerprints, not the original sequences.

Even when the does not depend on absolute hash map and replace it with Fuzzy hash map [32, 33] it still face difficulty to detect repackaged apps with small simple refactoring method

Schleimer et Al. [34] they attempt to find plagiarism with modifications using k-grams, by finding common token substrings of length k. If the differences between the programs are relatively infrequent or tend to be greater than k tokens apart then the comparison, will find many k-length token streams in common.

This approach also has a problem because insertion more than k instruction - even when those instruction are naïve and does not modify any behavior or flow- this approach will be failed to detect relation between the produced malware and the original one.

Unfortunately, even when these techniques has result better than feature based, it still vulnerable to addition or deletion of byte code instructions.

2.3.1.3 Program Dependency Graph (PDG) Based

In Apps There are two types of dependencies: data and control. Statement s1 has data dependency on statement s2 if s1 contains variable v, which v value changed in s2. On the other side, statement s1 has control dependency on statement s2 if s2 decide if s1 executed or not.

Crussell et Al. [35] working on detection clones of android apps, they exclude famous libraries as com.facebook.android and com.google.admob using sha1 hash to be sure these libraries untouched. After that, they create PDG for every method and apply losseless and lossy filters for every method pairs in the two apps. Lossless filter removes methods smaller than 10 nodes and lossy filter, which discards method pairs that are unlikely to match due to a difference in the distribution of types of nodes in the two PDGs. After that, they apply VF2

algorithm to compute subgraph isomorphisms. Finally, they calculate similarity of the two application and decide if these apps are clones or not.

This approach is good to determine clones but unfortunately, it has some back doors. First lossless filter can attacke by divide large method to smaller ones. They do not take noisy code in their account. Second it is good for clones but it takes too much time for malware detection and can`t find relation between malware and malicious app

Crussell et Al. in [36] they work on AnDarwin framework its design done on four stages: First, it represents each app as a set of vectors computed over the app's Program Dependence Graphs, split into connected components as multiple data-independent computations. Second, it finds similar code segments by clustering all the vectors of all apps. Third, it eliminates library code based on the frequency of the clusters. Finally, it detects apps that are similar, considering both full and partial app similarity.

CHEX [37] is a tool to detect component hijacking vulnerabilities in Android applications by tracking taints between externally accessible interfaces and sensitive sources or sinks. Although it does not built for the task, CHEX can be used for taint analysis. CHEX does not analyze calls into Android framework itself but instead requires a model of the framework. CHEX's entry-point model requires

an enumeration of all possible “split orderings”. Furthermore, CHEX is limited to at most 1-object-sensitivity.

LeakMiner [38] analyzes Android apps on market site. Thus, it does not introduce runtime overhead to normal execution of target apps. Besides, Leak Miner can detect information leakage before apps are distributed to users, it implements the Android lifecycle but the analysis is not context-sensitive - A context-sensitive analysis is an interprocedural analysis that considers the calling context when analyzing the target of a function call. In particular, using context information one can jump back to the original call site, whereas without that information, the analysis information has to be propagated back to all possible call sites, potentially losing precision-.

AndroidLeaks [39] also state the ability to handle the Android Lifecycle including callback methods. It is based on WALA’s context-sensitive System Dependence Graph with a context-insensitive overlay for heap tracking, but it taints the whole object if tainted data is stored in one of its fields, i.e., is neither field nor object sensitive. This precludes the precise analysis of many practical scenarios.

SCanDroid [40] is a tool for reasoning about data flows in Android applications. Its main focus is the inter-component (e.g. between two activities in

the same app) and inter-app data flow. This poses the challenge of connecting intent senders to their respective receivers in other applications. SCanDroid prunes all call edges to Android OS methods and conservatively assumes the base object, the parameters, and the return value to inherit taints from arguments.

EPICC [41] proposes a string analysis for inferring inter component communication specifications. These include inter component communication entry and exit points, information about the action, data and category components of intents used for inter component communication, as well as Intent key/value types.

2.4 Summary

There is many research efforts on Android malware detection, repackaging app detection, cloning apps detection and leakage information detection. They cover inter-process communication, permission up-used, information leakages and harmful operation but they do not discuss the idea of leakage information on multi app version. On our research, we will discuss this idea and create a tool to detect these leakages.

3- Research Tools

Malwares for Android application are considered as one of the most growing problems, so there must be a new techniques and tools to detect these malwares. In fact there are many antiviruses' tools in today market to detect malware using either static or dynamic analysis, In this chapter a scientific view will be presented for algorithms and techniques used for this research.

3.1 Reverse Engineering

Reverse Engineering is a process of analyzing program code or software in order to test it from any vulnerability or any errors. Reverse engineering is the ability to generate the source code from an executable code. This technique is used to examine the functioning of a program or to evade security bugs, etc. Reverse engineering can therefore be stated as a method or process of modifying a program in order to make it behave in a manner that the reverse engineer desires.

Joany Boutet has quoted Shwartz, saying,

"Whether it's rebuilding a car engine or diagramming a sentence, people can learn about many things simply by taking them apart and putting them back together again. That, in a nutshell, is the concept behind reverse engineering -breaking something down in order to understand it, build a copy or improve it " [42]

From the beginning of 2009, research scientists began proposes tools for reverse the DalvikBytecode. One of them is undX tool which could generate a JAR

file from an Android APK file, then convert to JAVA using tools such as JAD and and JD-GUI. The undX tool worked well with basic applications; but it posed many problems when dealing with complex Dalvik Bytecode. The Dex2Jar tool originated then. Dex2Jar does similar job to undX; but this tool also has some issues while dealing with complex Dalvik Bytecode.

The application, in its pre-compiled binary format, is distributed and hence it is not possible to directly debug the source code but there are disassemblers that convert or reverse the Dalvik Bytecode into readable format. The binaries for Dalvik Virtual Machines are in the .dex format. Backsmali [43] is a disassembler that is used for .dex files in Dalvik VM. Backsmali convert .dex file to intermediate language with full support of .dex and without lose anything.

3.2 Static Analysis

Static analyses inspect code to derive information about the application's behavior at runtime. Every application has variables (inputs from a user, files, internet etc.) an analysis has to abstract from concrete program runs. Static analyses aims to cover all possibilities by making assumptions. The properties derived from these assumptions can be weaker than the program's properties actually are, but they are guaranteed to be applicable for every program run. In this way, static analysis detects an application behavior, which might not actually

happen during runtime, but it does not miss a behavior, which can happen during runtime (i.e. privacy invasion).

In general, there are two different approaches to static analysis: type systems and data-flow based approaches. Type systems assign properties to components of the application and checks whether they are going to hold during run time. Data-flow based is a technique for gathering information about the possible set of values calculated at various points in an application.

Modern sophisticated tools convert the input (either bytecode or source code) to intermediate representations on which they can efficiently operate. To model the program flow they create control-flow graphs (CGF) and call graphs. CGF represent intra-procedural sequences of statements, call graphs contain edges between a call site and the call target.

Usually it is not possible to determine these targets unambiguously: The method invoked by the call site can refer to the implementation of the class specified in the call site or any other subclass. For example, a class A defines the method m() and has a subclass B. The call site x.m() can either refer to the implementation of A or B, depending on the initialization of x, which might not be statically resolvable.

3.2.1 Call graph (CF)

A call graph is a directed graph that represents calling relationships between functions in a computer program. Specifically, each node represents a function and each edge (f, g) indicates that function f calls function g. Thus, a cycle in the graph indicates recursive function calls.

Call graphs are a basic program analysis result that can be used for human understanding of programs, or as a basis for further analyses, such as an analysis that tracks the flow of values between functions. One simple application of call graphs is finding functions that are never called.

A static CG is a call graph intended to represent every possible run of the program. The exact static CG is an undecidable problem, so static call graph algorithms are generally over-approximations. That is, every call relationship that occurs is represented in the graph, and possibly some call relationships that would never occur in actual runs of the program. CG can be resource consumers in construction process, visiting call nodes and memory storage or example, constructing the CG of a Java “Hello, World!" program using Spark [44] can take up to 30 seconds, and produces a CG with 5,313 reachable methods and more than 23,000 edges. Because of that, CG can be defined to represent varying degrees of precision. A more precise CG more precisely approximates the behavior of the real program, at the cost of taking longer to compute and more memory to store. The

most precise CG is fully context-sensitive, which means that for each function, the graph contains a separate node for each call stack that function can be activated with. A fully context-sensitive CG is called calling context tree. A calling context tree can be computed dynamically easily, although it may take up a large amount of memory. Calling context trees are usually not computed statically, because it would take too long for a large program. The least precise call graph is context-insensitive, which means that there is only one node for each function. This is a tradeoff problem will be shown in the following example

```
public class Example1 {
    public static void main(String[] args) {
        String s1=newLine("hello");
        String s2=newLine("world");
        System.out.println(s1.concat(s2));
    }

    public static String newLine(String input)
{
        if(input.equals("hello"))
            return tab(input.concat("\n"));
        else
            return input.concat("\n");
    }

    public static String tab(String input) {
        return input.concat("\t");
```

```
    }
}
```

Code1 is simple java application contains three method to clarify context sensitivity levels no context sensitivity every method has only one node for method in simple words there is no different between call happened on tab call in newline and call happened on newline. On the other hand, context sensitive CG has node for every method call happened this gives more information about the context this method called on it.

3.3 WALA

Watson Libraries for Analysis (WALA) is a framework provides static and dynamic analysis capabilities for Java bytecode and related languages and for JavaScript. WALA is licensed under the Eclipse Public License. The initial WALA infrastructure was independently developed as part of the DOMO research project at the IBM T.J. Watson Research Center. In 2006, IBM donated the software to the community.

Core WALA Features

WALA features include:

1- Java type system and class hierarchy analysis

2- Source language framework supporting Java and JavaScript

3- Interprocedural dataflow analysis (RHS solver)

4- Context-sensitive tabulation-based slicer

5- Pointer analysis and call graph construction

   a. Several algorithms provided (RTA, variants of Andersen's analysis)

   b. Highly customizable (e.g., context sensitivity policy)

      i. ZeroCFA context insensitive

      ii. ZeroOneCFA context sensitive

   c. Tuned for performance (time and space)

6- Static single assignment form SSA-based register-transfer language IR

   a. SSA exsit on wala for Java and Java Script

   b. Anyone can extend it and IR for other languages

7- General framework for iterative dataflow

8- General analysis utilities and data structures

9- A bytecode instrumentation library (Shrike) and a dynamic load-time instrumentation library for Java (Dila).

10- Robustness, Efficiency and Extensibility.

3.4 Scandroid

SCanDroid [40] is a tool for reasoning about data flows in Android applications. Its focus is the inter-component (e.g. between two activities in the same app) and inter-app data flow. This poses the challenge of connecting intent

senders to their respective receivers in other applications. SCanDroid prunes all call edges to Android OS methods and conservatively assumes the base object, the parameters, and the return value to inherit taints from arguments.

Scandroid one of the first tools created to static analysis for android apps they tried to create automatic security certification for android application. SCANDROID's analysis is modular to allow incremental checking of applications as they are installed on an Android device. It extracts security specifications from manifests that accompany such applications, and checks whether data flows through those applications are consistent with those specifications.

They converted android byte code of delvaik VM to SSA-instruction compatible with intermediate representation (IR) form of WALA and make WALA framework able to use for static analysis for android applications. They make their source available on github.

We use their conversion of android byte code to IR, which is perfect, and used by almost all static analysis tools using WALA framework for android static analysis.

3.5 Summary

This chapter describes in theoretical view of the most important used tools in this thesis; First a scientifically description about static analysis and call graph

concepts has been discussed then a complete description about WALA framework with its usage and features has been provided and finally information about Scandroid and its implementation for android delvik byte code to WALA IR.

Those tools are used with reverse engineering as tool to insure results of these tools are precise and correct.

4- Methodology Evaluation and Analysis

In this chapter a methodology, experiments and thesis proposed algorithm will be discussed. In Section 5.1 discuss our proposed algorithm named Distributed Malware Detection Algorithm we create to solve leakage information over versions of application, Section 5.2 discuss implementation and used entry points, sources, sinks, transient sinks and transient sources , Section 5.3 is the experiment and evaluation, . Finally, section 5.4 is Summary for chapter.

4.1 DMDA Algorithm

In this thesis, we propose a new method to detect malwares distributed over application versions. Versioning will help malwares, which are leak information - Appendix A have example- and malwares can be divided to steps. The algorithm proposed will help to detect these malwares using call graph and pointer analysis.

4.1.1 Definitions

Definition 1 (Entry Point)

An Entry point is the point where operating system enters a program. In many programming languages, the main function is where a program starts its execution. Android is operating system with multiple entry point. Activity onCreate method is entry point.

Definition 2 (Source)

Source is calls into resource method returning non-constant value into the application code. This value is valuable to user privacy or user life. Example getDeviceId() resource method is an Android source. It returns a value (the IMEI) into the application code.

Definition 3 (Sink).

Sink is calls into resource method accepting at least one non-constant data value from the application code as parameter, if and only if those parameters go out the application. The sendTextMessage() resource method is an Android sink as the message text are possibly non-constant and goes to phone number.

Definition 4 (Transient Source)

Transient Source is calls into resource method returning non-constant value stored into local storage to the application code. This value is valuable to user privacy or user life. Example retrieve data from local database.

Definition 5 (Transient Sink)

Transient Sink is calls into resource method accepting one non-constant data value from the application code as parameter if and only if those parameters goes to local storage resource. Example saving contacts information on local database.

Definition 6 (Leak)

Leak is a call graph path where start in Source resource and end to Sink resource. Example Application send contacts data to internet website.

Definition 7 (Transient Leak)

Transient leak is a call graph path where start in Source resource and end to transient Sink resource. Example Application save contacts data into local storage media.

4.1.2 Attack model (Distributed Malware Attack Model)

Android is an open environment for development but this make it a field for malwares as demonstrated by [22 ،23 ،15]. Those researchers talked about misuse permissions, Exploiting over permissions by malicious applications and data leaks. Many researcher efforts talking about it as mentioned on chapter 2. Their efforts is focused on one version of android application. But android OS and many of android markets including Google Play market support versioning on android application [45] through android-mainfest.xml attributes android:versionCode and

android:versionName so it is simple to malware producer to distribute its malware on multi-version of android application. Version one got the data from android OS ex. contacts and SMSs and stored it on its own data, and on version two remove the code was responsible to store these data and add other code which is misuse these information like leak these data to internet.

4.1.3 DMDA Algorithm

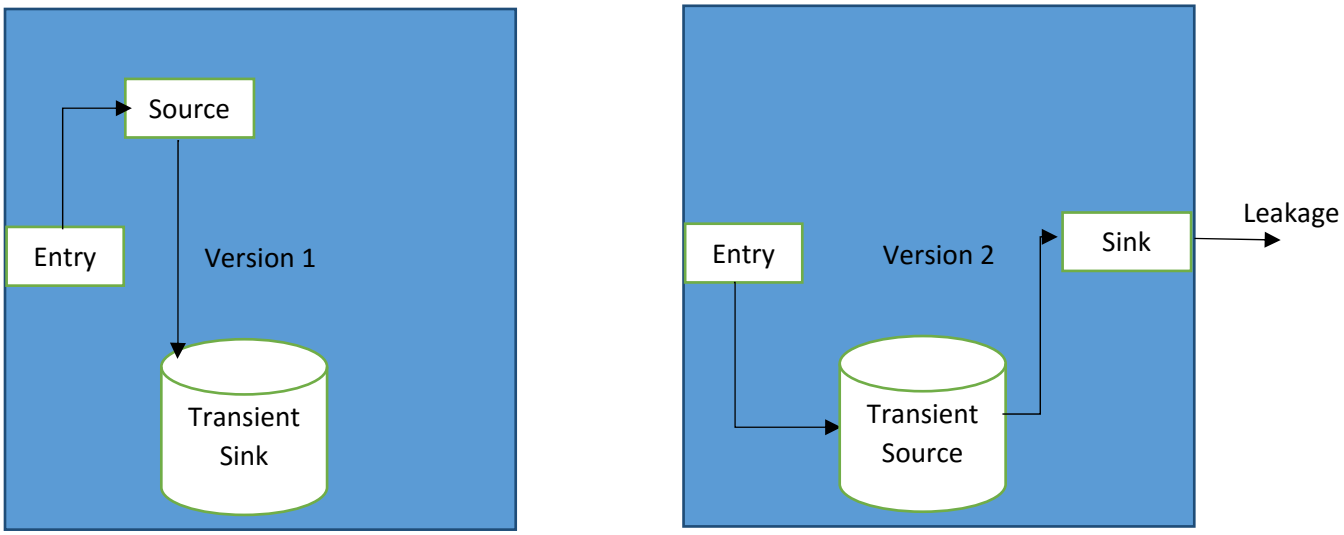

Figure 3: Attack Model

In the section we describe DMDA algorithm created to detect, malware apps distribute their malware behavior on their versions So these application split their Source and sink methods into two versions or more using transient sink and transient source.

Figure 3: Attack Model shows attack model, DMDA algorithm design to detect. This model distributed its source and sink into two different versions of the app.

DMDA algorithm steps are

1- Determine entry points of app version
2- Create call graph cg based on S where S is group of E and E is an entry point. Cg will contains N where N is a group of nodes and D where D is a group of edges
3- Visit call graph and determine PI –pure sink- , PO –pure source- , transient TI –transient sink- and TO –transient Source- nodes these nodes.
4- Using [47] to solve dependencies and reduce reachability.
5- if v == 1 then where v is the version of app
    a. call findLeak
    b. call findTransientSink
6- if transient think exist then
    a. call savetransientSink
7- if v>1
    a. call findTransientLeak

b. call findLeak

c. call findTransientSink

d. saveTransientSink

findLeak procedure

1- if there is path between source and sink then leak is exist and this is a malware app

saveTransientSink procedure

1- after finding transient sink check for possible paths used for this sink if there is path to source this path will saved
2- save the key used to this parameter –example table name for inset statement -

findTransientLeak procedure

1- if transient sink saved before have the same key of transient sink then this transient leak do
    a. replace transient sink with transient source paths stored before
2- else
    a. ignore this transient source

Figure 4: DDMA Algorithm shows the steps of DMDA algorithm in a simple way represent the model present on attack model in Figure 3: Attack Model and shows main idea of distribution and where main steps of algorithm happened.

Figure 5: DDMA Algorithm`s Flow shows steps of DMDA algorithm in simple flowchart diagram. The diagram focused on main steps of algorithm like transient sink and transient source.

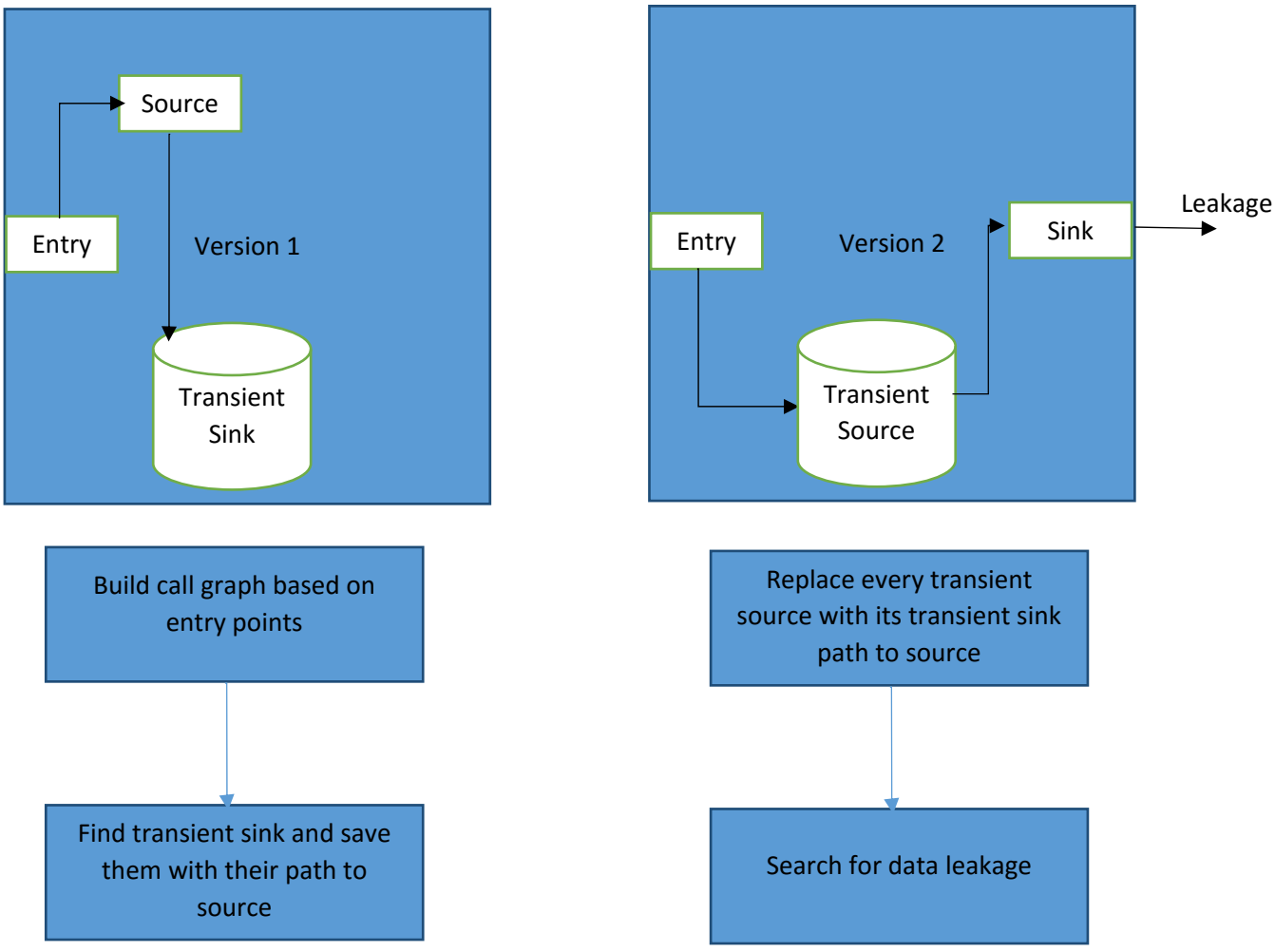

Figure 4: DDMA Algorithm`s Model

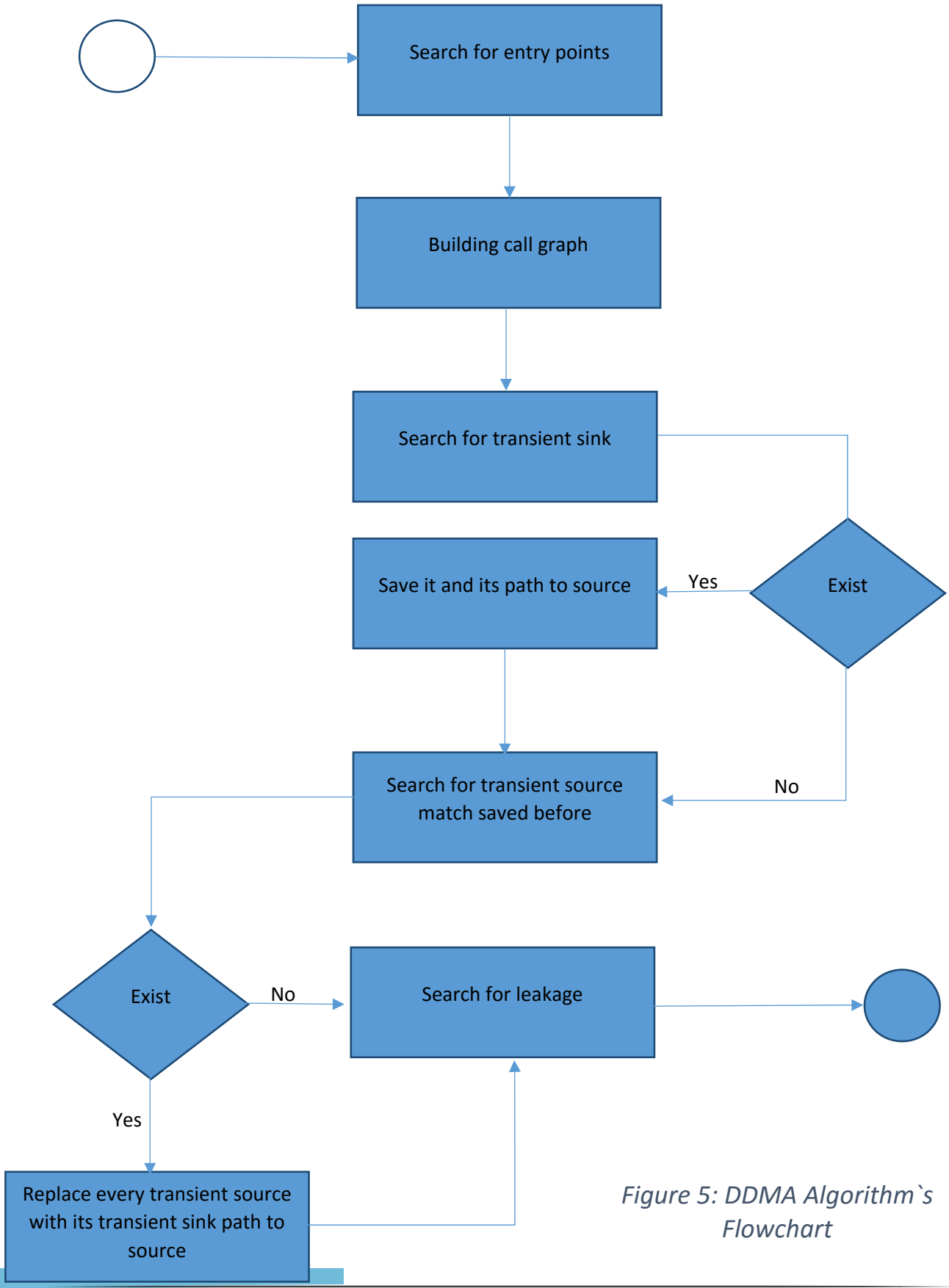

Figure 5: DDMA Algorithm`s Flowchart

4.2 Implementation

This chapter explain DMDA algorithm implementation, entry points for android application, sources, sinks, transient sinks and transient sources and how WALA frameworks used to implement DMDA algorithm. The real code attached on Appendix B to more details.

4.2.1 Android Entry points

Android is not like java android have multi-entry point those entry points are the real start of building android app call graph. Entry points of android are:

1- Android.app.Activity: onCreate, onStart, onRestart, onResume, onPause, onStop, onDestroy, onActivityResult, onRestoreInstanceState, onSaveInstanceState.

2- Android.app.Service: onCreate, onStart, onStartCommand, onDestroy, onBind.

3- ContentProvider: onCreate, query, insert, update, delete.

4- BroadcastReceiver: onReceive.

Those are the entry points Android OS can start android application from them and those are the seed of call graph.

4.2.2 Source and Sink

After determine entry points, must determine source and sink of our analysis. Source and sink concept is a famous concept in privacy and data leak analysis where data is a precious. Source is a method return a valuable data like phone number, user SMSs, contacts, browser history etc…. Sink is a method leak these data or misuse like send it over internet, SMS, Bluetooth etc…

For this research, we choose these Sources

1- Android.content.ContentResolver. query: this method used to query any content provider on android app. It is one of the most famous sources on android. Any developer with the right permission can query SMS, Contacts, browsing history and other app data.
2- Android.location.LocationManager.[all methods]: this class is responsible to location stuff contains GPS providers and location and last known plcae.
3- Android.telephony.TelephonyManager.[getNeighboringCellInfo| getCellLocation]: these methods return data about GSM cells these information can leak user location.

In addition, these Sinks:

1- Android.app.Activity.setResult: this method used to respond on call of startActivityForResult and it can leak data on it is parameter to other android application.

2- Android.app.Activity.[starActivity| starActivityForResult| startActivityIfNeeded| startNextMatchingActivity| startActivityFromChild]: these methods can leak data to other application on the intent send to start their activities.
3- Android.content.ContentResolver.[query|insert|update|delete]: these methods help developers to access Content Provider query, insert, update and delete
4- Android.telephony.gsm.SmsManager.[sendTextMessage|sendDataMessage| sendMultipartTextMessage]: those methods can leak data through GSM messages.
5- Android.net.AndroidHttpClient.execute: this method can leak data through it is parameter to internet.
6- java.net.HttpURLConnection.[getOutputStream| setRequestProperty]:these methods can leak data through http request on header or body.
7- java.net.CookieManager. setCookie: this method can leak data through http header called cookies.

Those are not all the sources and sinks on android those are the ones used on our research to prove the idea there is other research doing hard work on this point [24, 19, 40].

4.2.3 Transient Sources and Sinks

Transient sources and sinks are those methods used to store application data into local storage. Those cannot consider as a pure sources and pure sinks because they almost used to store clear data so considering them as a source or sink probably result a false positive malware detection. But ignoring them lead to miss malwares divided into application versions.

For this research, we choose the following method as transient Sink:

1- android.content.SharedPreferences.Editor[putBoolean| putFloat| putInt| putLong| putString| putStringSet]: shared preference used to persistent primitive data or Strings and reuse them after a while. These methods can transiently leak data through their second parameter.
2- android.database.sqlite.SQLiteDatabase[insert| insertOrThrow| insertWithOnConflict| replace| replaceOrThrow| update| updateWithOnConflict]:SQLLite is a simple relational database used to store complex data types and reuse them with fast query. These methods can transiently leak data through their second parameter through their parameter ContentValues.

For this research, we choose the following method as transient Source:

1- android.content.SharedPreferences [getBoolean| getFloat| getInt| getLong| getString| getStringSet]: these methods used to retrieve data stored on put methods. These methods can transiently been a source of data through their return values.

2- android.database.sqlite.SQLiteDatabase[query| queryWithFactory| rawQuery| rawQueryWithFactory]: those methods used to retrieve data stored using update, insert and replace methods. These methods can transiently been a source of data through their return values.

All these lists –sources, sinks, transient sinks, transient sources and entry points- included on eAndroidSPec.java, which is, extend ISpec class one of scandroid specifications.

4.2.4 Exclusion list

As described on this research call graph and static analysis is greedy for memory even simple ones can take too much memory. Because of that WALA have exclusion list, which used to exclude unimportant classes from call graph and data flow analysis. We exclude famous used libraries and basic java packages. This technique help WALA to reduce memory and increase productivity. The full list of excluded packages in Appendix B.

4.2.5 Implementation

Using WALA Framework help in implementation a lot of algorithm implementing and available to extend and reuse. We use WALA call graph which depends on graph reachability concept and Pointer analysis implementation using kidall`s Framework [49] to follow keys of transient source and transient sinks.

Kidall`s Framework based on simple constant propagation this idea was created firstly by Kidall in [49] to Discover values that are constant on all possible executions, and propagate values.

Simply this algorithm steps are start on entry point, process this entry point and produce constant propagation, send these information to all first successor of this entry point, repeat this in next successors, merge these information –intersection them- if the data on variable is different on two branches this variable are not included on data return by the algorithm.

This algorithm used into extract keys used to store data in transient sink and transient source.

Based on these algorithms we build two filters: Leakage Filter, which responsible to detect leakage malware behavior and transient filter, which responsible to detect transient leakage and replace transient nodes with the original code.

Transient leakage filter build call graph with context sensitivity for string objects after building call graph this filter search for transient source and check previous list of transient sink if the key of transient source equal one of the these keys it replace the statement with call graph saved for that key finally the filter search for the transient sink and store them with their pruned call graph.

Leakage filter take the call graph built in transient leakage filter, start searching of paths connect source, and sink if there is a path or more then this filter recognize this app as a malware.

4.3 Experiment: Malware detection

In this section, two experiments made to show and explain the attack and experiment effectiveness of DDMA Algorithm. First experiment focus on the attack model and how distribution of a famous malware in two versions make most of anti-malware blind. The second experiment check effectiveness of DMDA algorithm to find malware behavior distributed over android app versions and check these apps.

4.3.1 Attack Model Expirment

Therefore, we think any malware distributed on app versions make most of anti-malware blind even for simple, old and famous malwares like DroidKungFu [46]. DroidKungFu is a Trojan, which although seemingly inoffensive, can actually carry out attacks and intrusions: screen logging, stealing personal data, etc. We use

DroidKungFu as example to explain the attack model. Appendix A contains the DroidKungFu on two versions we test the two versions on VirusTotal –which is a free online service subsidiary of Google that analyzes files and URLs enabling the identification of viruses, worms, trojans and other kinds of malicious content detected by antivirus engines and website scanners – and no anti-malware of 57 scanning the APKs catch the malware.

4.3.2 Effectiveness of DDMA

For this experiments chosen group of apps include the app in Appendix A with two versions of every app these apps taken from [50].

This group contains 100 apps all of them related to contact APIs for every app two chosen versions in this sample next table show the results

No. of apps	No. of versions for app	Transient sources	Transient sinks	sources	leakages
100	2	156	209	200	2

We find over 200 transient sinks and over 150 transient sources these are not a leakage but these may turn on future to leakage. We find also two leakages.

We check the versions where leakage happened by using reverse engineering tools discussed before what we found is interesting one is contains the problem the other one was false positive because of SQL complications.

Select mydata from ComplexMyDataAndContactsData;

This was the reason of false positive we have. We will discuss this issue in future work.

Also we got attention that some apps stored unimportant data like contact id or contact created time these will be considered on our application as a leakage but these values does not have any valued so developers may send it without mean to leakage data.

5- Conclusion and Future work

Today life activities for all people depend on latest technologies, which provide fast and available communication, and production services, smartphones are one of those technologies, it is used everywhere, by everyone, for almost purposes. The wide use of smartphones applications leads for wide growth in Malwares applications, which aims, to threat users.

Android is the most shared OS for smart phones and it has the biggest number of malwares. In this thesis an Introduction about Android has been discussed from Android website we talked about Android architecture , components and activity life cycle; Thesis talked about Malwares in general and Malware in Android applications with more details, In addition this thesis summarized a group of related work in the topic of Android Malwares detection and leakage detection.

Our contributing was to detect the malware behavior specially leakage data on app versions. The research main idea is to find transient source and transient sink and convert them to their original call graph which help solving malware distribution.

We used call graph to determine reachability and kidall`s Framework to solve dependencies and determine transient sources and sinks.

To evaluate our idea we tested group of apps on our experiment. We find over 200 transient sinks and over 150 transient sources these are not a leakage but these may turn on future to leakage. We find also two leakages.

As a future work enlarge the dataset by immigrate it with other existing malwares datasets will decrease the false detection. Also need to create SQL parser to exclude false positive on these cases query on the same table but on local app data not on data stored from other data provider and also data does not have any value like contact id and contact created time.

References

[1] R. Sharp, An Introduction to Malware, Technical University of Denmark, 2013.

[2] B. Solvar and P. S. Rene, "Privacy services for mobile devices," 2011.

[3] W. Enck, M. Ongtang and P. McDaniel, "Privacy services for mobile devices," *IEEE Security & Privacy Magazine,* 2009.

[4] W. Enck, D. Octeau, P. McDaniel and S. Chaudhuri, "A Study of Android Application Security," in *USENIX Security*, 2011.

[5] K. Hamandi, A. Chehab, I. Elhajj and A. Kayssi, "Android SMS Malware: Vulnerability and Mitigation," in *International Conference on Advanced Information Networking and Applications*, 2013.

[6] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *IEEE Symposium on Security and Privacy*, 2012.

[7] "Android Developers," Google, 2014. [Online]. Available: http://developer.android.com/guide/topics/data/data-storage.html.

[8] "Anatomy Physiology of an Android," Google, 2008. [Online]. Available: http://androidteam.googlecode.com/files/Anatomy-Physiology-of-an-Android.pdf.

[9] "Activities," Google, [Online]. Available: http://developer.android.com/guide/components/activities.html. [Accessed 12 2014].

[10] "Activity," Google, [Online]. Available: http://developer.android.com/reference/android/app/Activity.html.

[11] "SQLiteOpenHelper," Google, [Online]. Available: http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html.

[12] T. Isohara, "Kernel-based Behavior Analysis for Android Malware Detection," in *IEEE Seventh International Conference on Computational ntelligence and Security*, 2011.

[13] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *Security and Privacy (SP), IEEE Symposium*, 2012.

[14] W. Zhou, Y. Zhou, X. Jiang and P. Ning, "Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces," *ACM,* 2012.

[15] M. Grace, Y. Zhou, Z. Wang and X. Jia, "Systematic Detection of Capability Leaks in Stock Android Smartphones," in *19th NDSS*, 2012.

[16] Y. Zhou, Z. Wang, W. Zhou and X. Jiang, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets," in *NDSS*, 2012.

[17] "Android and Security," Google, 2012. [Online]. Available: http://googlemobile.blogspot.com/2012/02/android-and-security.html. [Accessed 2014].

[18] W. Enck, M. Ongtang and P. McDaniel, "On lightweight mobile phone application certification," in *ACM conference on Computer and communications security*, 2009.

[19] W. Enck, D. Octeau, P. McDaniel and S. Chaudhuri, "A study of android application security," in *USENIX conference on Security* , 2011.

[20] E. Chin, A. Porter Felt, K. Greenwood and D. Wagner, "Analyzing inter-application communication in Android," in *international conference on Mobile systems, applications, and services*, 2011.

[21] N. Hardy, "The Confused Deputy: (or why capabilities might have been invented)," in *ACM SIGOPS Operating Systems Review*, 1988.

[22] L. Davi, A. Dmitrienko, A.-R. Sadeghi and M. Winandy, "Privilege escalation attacks on android," in *international conference on Information security* , 2011.

[23] A. Felt, H. Wang, A. Moshchuk, S. Hanna and E. Chin, "Permission Re-Delegation: Attacks and Defenses," in *USENIX Security Symposium*, 2011.

[24] W. Enck, P. Gilbert, B.-G. Chun, L. Cox, J. Jung, P. McDaniel and A. Sheth, "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," in *9th USENIX Symposium on Operating Systems Design and Implementation*, 2011.

[25] Y.-C. Jhi, X. Wang, X. Jia, S. Zhu, P. Liu and D. Wu, "Value-Based Program Characterization and Its Application to Software Plagiarism Detection," in *Proceeding of the 33rd International Conference on Software Engineering*, 2011.

[26] G. Myles and C. Collberg, "Detecting Software Theft via Whole Program Path Birthmarks," *Information Security,* p. 404–415, 2004.

[27] T. Eder, M. Rodler, D. Vymazal and M. Zeilinger, "ANANAS – A Framework For Analyzing Android Applications," in *International Conference on Availability, Reliability and Security*, 2013.

[28] A. Reina, A. Fattori and L. Cavallaro, "A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors," in *EuroSec*, 2013.

[29] W. B. Tesfay, T. Booth and K. Andersson, "Reputation Based Security Model for Android Applications," in *IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012.

[30] V. Moonsamy, J. Rong and S. Liu, "Mining permission patterns for contrasting clean and malicious," *Elsevier,* 2013.

[31] H.-S. Ham and . M.-J. Choi, "Analysis of Android Malware Detection Performance using Machine Learning Classifiers," *IEEE,* 2013.

[32] "Rolling Hash (Rabin-Karp Algorithm)," Intro to Algorithms course at MIT.

[33] D. Hurlbut, "Fuzzy Hashing for Digital Forensic Investigators," 2009.

[34] "Winnowing: local algorithms for document fingerprinting," in *ACM SIGMOD International ACM SIGMOD International*, 2003.

[35] J. Crussell, C. Gibler and H. Chen, "Attack of the Clones: Detecting Cloned Applications on Android Markets," in *Springer-Verlag Berlin Heidelberg*, 2012.

[36] J. Crussell, C. Gibler and H. Chen, "AnDarwin: Scalable Detection of Semantically Similar Android Applications," in *Computer Security – ESORICS 2013*, 2013.

[37] L. Lu, Z. Li, Z. Wu, W. Lee and G. Jiang, "CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities," in *ACM conference on Computer and communications security*, 2012.

[38] Z. Yang and M. Yang, "LeakMiner: Detect Information Leakage on Android with Static Taint Analysis," *IEEE,* 2012.

[39] C. Gibler, J. Crussell, J. Erickson and H. Chen, "Scale, AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large," *Trust and Trustworthy Computing,* vol. 7344, pp. 291-307, 2012.

[40] A. Fuchs, A. Chaudhuri and J. Foster, "SCanDroid: Automated Security Certification of Android Applications," in *Proceedings of the 31st IEEE Symposium on Security and Privacy*, 2010.

[41] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein and Y. L. Traon, "Effective inter-component communication mapping in Android with Epicc: An essential step towards holistic security analysis," in *USENIX Security Symposium*, 2013.

[42] "Reverse Engineering Of Malware On Android," InfoSec, 2011.

[43] "smali - An assembler/disassembler for Android's dex format," [Online]. Available: https://code.google.com/p/smali/. [Accessed 2014].

[44] O. Lhoták and L. Hendren, "Scaling Java points-to analysis using SPARK," in *International Conference on Compiler Construction*, 2003.

[45] "Versioning Your Applications," Google, [Online]. Available: http://developer.android.com/tools/publishing/versioning.html. [Accessed 2015].

[46] "8 Notorious Android Malware Attacks," [Online]. Available: http://www.informationweek.com/mobile/8-notorious-android-malware-attacks/d/d-id/1099385.

[47] T. Reps, S. Horwitz and M. Sagiv, "Precise interprocedural dataflow analysis via graph reachability," in *22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1995.

[48] T. Reps, S. Horwitz and M. Sagiv, "Precise Interprocedural Dataflow Analysis via Graph Reachability," in *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1995.

[49] G. Kildall, "A Unified Approach to Global Program Optimization," in *Proceedings of the 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 1973.

[50] "F-Droid," [Online]. Available: https://f-droid.org/repository/browse/. [Accessed 1/12/2014].

Appendices

Appendix A

Version 1

MainActivity.java

```
package com.example.droidkunfu;

import java.io.FileOutputStream;

import com.android.volley.Request;
import com.android.volley.Response.ErrorListener;
import com.android.volley.Response.Listener;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;

import android.support.v7.app.ActionBarActivity;
import android.content.Context;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends ActionBarActivity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);
		String mImei = Util.PhoneState.getImei(this);
		String mModel = Util.PhoneState.getModel();
		mModel = mModel.replaceAll(" ", "_");
		String mOsType = Util.PhoneState.getSDKVersion()[0];
		mOsType = mOsType.replaceAll(" ", "_");
		String mOsAPI = Util.PhoneState.getSDKVersion()[1];
		mOsAPI = mOsAPI.replaceAll(" ", "_");
		String string = mImei + " " + mModel + " " + mOsType + " " + mOsAPI;
		SharedPreferences preferences = getSharedPreferences("test",
				Context.MODE_PRIVATE);
		Editor editor = preferences.edit();
		editor.putString("message", string);
		editor.commit();
	}
}
```

Android-mainfest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.droidkunfu"
    android:versionCode="1"
```

```
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="9"
        android:targetSdkVersion="21" />

    <uses-permission android:name="android.permission.READ_PHONE_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Version 2

MainActivity.java

```
package com.example.droidkunfu;

import java.io.FileOutputStream;

import com.android.volley.Request;
import com.android.volley.Response.ErrorListener;
import com.android.volley.Response.Listener;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;

import android.support.v7.app.ActionBarActivity;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends ActionBarActivity {

      @Override
      protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            String message = preferences.getString("message", null);
```

```
            StringRequest request = new StringRequest(Request.Method.POST,
                        "http://iugaza.edu.ps?test=" + message, new
Listener<String>() {

                              @Override
                              public void onResponse(String arg0) {
                                    // TODO Auto-generated method stub

                              }
                        }, new ErrorListener() {

                              @Override
                              public void onErrorResponse(VolleyError arg0) {
                                    // TODO Auto-generated method stub

                              }
                        });
            Volley.newRequestQueue(this).add(request);
      }
}
```

Android-mainfest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.droidkunfu"
    android:versionCode="2"
    android:versionName="2.0" >

    <uses-sdk
        android:minSdkVersion="9"
        android:targetSdkVersion="21" />

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Appendix B

Java60RegressionExclusions.txt

```
android\/accessibilityservice.*
android\/accounts.*
android\/animation.*
android\/annotation.*
android\/app\/admin.*
android\/app\/backup.*
android\/app\/job.*
android\/app\/usage.*
android\/appwidget.*
android\/bluetooth.*
android\/bluetooth\/le.*
android\/content\/pm.*
android\/content\/res.*
android\/database.*
android\/database\/sqlite.*
android\/drm.*
android\/gesture.*
android\/graphics.*
android\/graphics\/drawable.*
android\/graphics\/drawable\/shapes.*
android\/graphics\/pdf.*
android\/hardware.*
android\/hardware\/camera2.*
android\/hardware\/camera2\/params.*
android\/hardware\/display.*
android\/hardware\/input.*
android\/hardware\/usb.*
android\/inputmethodservice.*
android\/location.*
android\/media.*
android\/media\/audiofx.*
android\/media\/browse.*
android\/media\/effect.*
android\/media\/projection.*
android\/media\/session.*
android\/media\/tv.*
android\/mtp.*
android\/net.*
android\/net\/http.*
android\/net\/nsd.*
android\/net\/rtp.*
android\/net\/sip.*
android\/net\/wifi.*
android\/net\/wifi\/p2p.*
android\/net\/wifi\/p2p\/nsd.*
android\/nfc.*
android\/nfc\/cardemulation.*
android\/nfc\/tech.*
android\/opengl.*
```

```
android\/os\/storage.*
android\/preference.*
android\/print.*
android\/print\/pdf.*
android\/printservice.*
android\/provider.*
android\/renderscript.*
android\/sax.*
android\/security.*
android\/service\/dreams.*
android\/service\/media.*
android\/service\/notification.*
android\/service\/restrictions.*
android\/service\/textservice.*
android\/service\/voice.*
android\/service\/wallpaper.*
android\/speech.*
android\/speech\/tts.*
android\/support\/annotation.*
android\/support\/multidex.*
android\/support\/v17\/leanback.*
android\/support\/v17\/leanback\/app.*
android\/support\/v17\/leanback\/database.*
android\/support\/v17\/leanback\/graphics.*
android\/support\/v17\/leanback\/widget.*
android\/support\/v4\/accessibilityservice.*
android\/support\/v4\/content\/pm.*
android\/support\/v4\/content\/res.*
android\/support\/v4\/database.*
android\/support\/v4\/graphics.*
android\/support\/v4\/graphics\/drawable.*
android\/support\/v4\/hardware\/display.*
android\/support\/v4\/media.*
android\/support\/v4\/media\/session.*
android\/support\/v4\/net.*
android\/support\/v4\/print.*
android\/support\/v4\/provider.*
android\/support\/v4\/text.*
android\/support\/v4\/util.*
android\/support\/v4\/view\/accessibility.*
android\/support\/v7\/appcompat.*
android\/support\/v7\/cardview.*
android\/support\/v7\/graphics.*
android\/support\/v7\/gridlayout.*
android\/support\/v7\/media.*
android\/support\/v7\/mediarouter.*
android\/support\/v8\/renderscript.*
android\/system.*
android\/telecom.*
android\/telephony.*
android\/telephony\/cdma.*
android\/telephony\/gsm.*
android\/test.*
android\/test\/mock.*
```

```
android\/test\/suitebuilder.*
android\/test\/suitebuilder\/annotation.*
android\/text.*
android\/text\/format.*
android\/text\/method.*
android\/text\/style.*
android\/text\/util.*
android\/transition.*
android\/util.*
android\/view\/accessibility.*
android\/view\/animation.*
android\/view\/inputmethod.*
android\/view\/textservice.*
android\/webkit.*
com\/android\/internal\/backup.*
com\/android\/internal\/os.*
com\/android\/internal\/statusbar.*
com\/android\/internal\/widget.*
com\/android\/test\/runner.*
dalvik\/annotation.*
dalvik\/bytecode.*
dalvik\/system.*
java\/awt\/font.*
java\/beans.*
java\/lang\/annotation.*
java\/lang\/ref.*
java\/lang\/reflect.*
java\/math.*
java\/net.*
java\/nio.*
java\/nio\/channels.*
java\/nio\/channels\/spi.*
java\/nio\/charset.*
java\/nio\/charset\/spi.*
java\/security.*
java\/security\/acl.*
java\/security\/cert.*
java\/security\/interfaces.*
java\/security\/spec.*
java\/sql.*
java\/text.*
java\/util.*
java\/util\/concurrent.*
java\/util\/concurrent\/atomic.*
java\/util\/concurrent\/locks.*
java\/util\/jar.*
java\/util\/logging.*
java\/util\/prefs.*
java\/util\/regex.*
java\/util\/zip.*
javax\/crypto.*
javax\/crypto\/interfaces.*
javax\/crypto\/spec.*
javax\/microedition\/khronos\/egl.*
```

```
javax\/microedition\/khronos\/opengles.*
javax\/net.*
javax\/net\/ssl.*
javax\/security\/auth.*
javax\/security\/auth\/callback.*
javax\/security\/auth\/login.*
javax\/security\/auth\/x500.*
javax\/security\/cert.*
javax\/sql.*
javax\/xml.*
javax\/xml\/datatype.*
javax\/xml\/namespace.*
javax\/xml\/parsers.*
javax\/xml\/transform.*
javax\/xml\/transform\/dom.*
javax\/xml\/transform\/sax.*
javax\/xml\/transform\/stream.*
javax\/xml\/validation.*
javax\/xml\/xpath.*
junit\/framework.*
junit\/runner.*
org\/apache\/http.*
org\/apache\/http\/auth.*
org\/apache\/http\/auth\/params.*
org\/apache\/http\/client.*
org\/apache\/http\/client\/entity.*
org\/apache\/http\/client\/methods.*
org\/apache\/http\/client\/params.*
org\/apache\/http\/client\/protocol.*
org\/apache\/http\/client\/utils.*
org\/apache\/http\/conn.*
org\/apache\/http\/conn\/params.*
org\/apache\/http\/conn\/routing.*
org\/apache\/http\/conn\/scheme.*
org\/apache\/http\/conn\/ssl.*
org\/apache\/http\/conn\/util.*
org\/apache\/http\/cookie.*
org\/apache\/http\/cookie\/params.*
org\/apache\/http\/entity.*
org\/apache\/http\/impl.*
org\/apache\/http\/impl\/auth.*
org\/apache\/http\/impl\/client.*
org\/apache\/http\/impl\/conn.*
org\/apache\/http\/impl\/conn\/tsccm.*
org\/apache\/http\/impl\/cookie.*
org\/apache\/http\/impl\/entity.*
org\/apache\/http\/impl\/io.*
org\/apache\/http\/io.*
org\/apache\/http\/message.*
org\/apache\/http\/params.*
org\/apache\/http\/protocol.*
org\/apache\/http\/util.*
org\/json.*
org\/w3c\/dom.*
```

```
org\/w3c\/dom\/ls.*
org\/xml\/sax.*
org\/xml\/sax\/ext.*
org\/xml\/sax\/helpers.*
org\/xmlpull\/v1.*
org\/xmlpull\/v1\/sax2.*
```

AndroidSPec.java

```
package org.distibuteme;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.scandroid.spec.CallArgSinkSpec;
import org.scandroid.spec.CallRetSourceSpec;
import org.scandroid.spec.ISpecs;
import org.scandroid.spec.MethodNamePattern;
import org.scandroid.spec.SinkSpec;
import org.scandroid.spec.SourceSpec;
import org.scandroid.util.LoaderUtils;

import com.ibm.wala.classLoader.IClass;
import com.ibm.wala.classLoader.IMethod;
import com.ibm.wala.ipa.cha.ClassHierarchy;
import com.ibm.wala.types.ClassLoaderReference;

public class AndroidSpecs implements ISpecs {

	static String act = "Landroid/app/Activity";
	static String svc = "Landroid/app/Service";
	static String prv = "Landroid/content/ContentProvider";
	static String brc = "Landroid/content/BroadcastReceiver";
	static String rslv = "Landroid/content/ContentResolver";

	static String http = "Landroid/net/AndroidHttpClient";
	static String lm = "Landroid/location/LocationManager";
	static String tm = "Landroid/telephony/TelephonyManager";
	static String smsGsm = "android/telephony/gsm/SmsManager";
	static String ll = "Landroid/location/LocationListener";
	static String httpURL = "Ljava/net/HttpURLConnection";

	static String cookie = "Ljava/net/CookieManager";

	static String shared = "Landroid/content/SharedPreferences";
	static String sharedEditor = "Landroid/content/SharedPreferences/Editor";
	static String database = "Landroid/database/sqlite/SQLiteDatabase";

	static MethodNamePattern actCreate = new MethodNamePattern(act, "onCreate");
	static MethodNamePattern actStart = new MethodNamePattern(act, "onStart");
	static MethodNamePattern actResume = new MethodNamePattern(act, "onResume");
	static MethodNamePattern actStop = new MethodNamePattern(act, "onStop");
```

```
static MethodNamePattern actRestart = new MethodNamePattern(act,
            "onRestart");
static MethodNamePattern actDestroy = new MethodNamePattern(act,
            "onDestroy");
static MethodNamePattern actOnActivityResult = new MethodNamePattern(act,
            "onActivityResult");

static MethodNamePattern brcReceive = new MethodNamePattern(brc,
            "onReceive");

static MethodNamePattern actSetResult = new MethodNamePattern(act,
            "setResult");

static MethodNamePattern actStartActivityForResult = new MethodNamePattern(
            act, "startActivityForResult");
static MethodNamePattern actStartActivityIfNeeded = new MethodNamePattern(
            act, "startActivityIfNeeded");
static MethodNamePattern actStartNextMatchingActivity = new MethodNamePattern(
            act, "startNextMatchingActivity");
static MethodNamePattern actStartActivityFromChild = new MethodNamePattern(
            act, "startActivityFromChild");

static MethodNamePattern svcCreate = new MethodNamePattern(svc, "onCreate");
static MethodNamePattern svcStart = new MethodNamePattern(svc, "onStart");
static MethodNamePattern svcStartCommand = new MethodNamePattern(svc,
            "onStartCommand");
static MethodNamePattern svcBind = new MethodNamePattern(svc, "onBind");
static MethodNamePattern svcDestroy = new MethodNamePattern(svc,
            "onDestroy");

static MethodNamePattern rslvQuery = new MethodNamePattern(rslv, "query");
static MethodNamePattern rslvInsert = new MethodNamePattern(rslv, "insert");
static MethodNamePattern rslvUpdate = new MethodNamePattern(rslv, "update");
static MethodNamePattern rslvDelete = new MethodNamePattern(rslv, "delete");

static MethodNamePattern prvCreate = new MethodNamePattern(prv, "onCreate");
static MethodNamePattern prvQuery = new MethodNamePattern(prv, "query");
static MethodNamePattern prvInsert = new MethodNamePattern(prv, "insert");
static MethodNamePattern prvUpdate = new MethodNamePattern(prv, "update");
static MethodNamePattern prvDelete = new MethodNamePattern(prv, "delete");

static MethodNamePattern httpExecute = new MethodNamePattern(http,
            "execute");
static MethodNamePattern httpURLGetOutputStream = new MethodNamePattern(
            httpURL, "getOutputStream");
static MethodNamePattern httpURLSetRequestProperty = new MethodNamePattern(
            httpURL, "getOutputStream");
static MethodNamePattern cookieSetCookie = new MethodNamePattern(cookie,
            "getOutputStream");

static MethodNamePattern putBooleanShared = new MethodNamePattern(
            sharedEditor, "putBoolean");
static MethodNamePattern putFloatShared = new MethodNamePattern(
            sharedEditor, "putFloat");
```

```
static MethodNamePattern putIntShared = new MethodNamePattern(sharedEditor,
        "putInt");
static MethodNamePattern putLongShared = new MethodNamePattern(
        sharedEditor, "putLong");
static MethodNamePattern putStringShared = new MethodNamePattern(
        sharedEditor, "putString");
static MethodNamePattern putStringSetShared = new MethodNamePattern(
        sharedEditor, "putStringSet");
static MethodNamePattern getBooleanShared = new MethodNamePattern(shared,
        "getBoolean");
static MethodNamePattern getFloatShared = new MethodNamePattern(shared,
        "getFloat");
static MethodNamePattern getIntShared = new MethodNamePattern(shared,
        "getInt");
static MethodNamePattern getLongShared = new MethodNamePattern(shared,
        "getLong");
static MethodNamePattern getStringShared = new MethodNamePattern(shared,
        "getString");
static MethodNamePattern getStringSetShared = new MethodNamePattern(shared,
        "getStringSet");

static MethodNamePattern insertDatabase = new MethodNamePattern(database,
        "insert");
static MethodNamePattern insertOrThrowDatabase = new MethodNamePattern(
        database, "insertOrThrow");
static MethodNamePattern insertWithOnConflictDatabase = new MethodNamePattern(
        database, "insertWithOnConflict");
static MethodNamePattern replaceDatabase = new MethodNamePattern(database,
        "replace");
static MethodNamePattern replaceOrThrowDatabase = new MethodNamePattern(
        database, "replaceOrThrow");
static MethodNamePattern updateDatabase = new MethodNamePattern(database,
        "insertOrThrow");
static MethodNamePattern updateWithOnConflictDatabase = new MethodNamePattern(
        database, "updateWithOnConflict");
static MethodNamePattern queryDatabase = new MethodNamePattern(database,
        "query");
static MethodNamePattern queryWithFactoryDatabase = new MethodNamePattern(
        database, "queryWithFactory");
static MethodNamePattern rawQueryDatabase = new MethodNamePattern(database,
        "rawQuery");
static MethodNamePattern rawQueryWithFactoryDatabase = new MethodNamePattern(
        database, "rawQueryWithFactory");

static MethodNamePattern llLocChanged = new MethodNamePattern(ll,
        "onLocationChanged");
static MethodNamePattern llProvDisabled = new MethodNamePattern(ll,
        "onProviderDisabled");
static MethodNamePattern llProvEnabled = new MethodNamePattern(ll,
        "onProviderEnabled");
static MethodNamePattern llStatusChanged = new MethodNamePattern(ll,
        "onStatusChanged");

private static MethodNamePattern[] defaultCallbacks = { actCreate,
```

```
                actStart, actResume, actStop, actRestart, actDestroy,
                actOnActivityResult,

                svcCreate, svcStart, svcStartCommand, svcBind, svcDestroy,

                prvCreate, prvQuery, prvInsert, prvUpdate, brcReceive

        };

        public MethodNamePattern[] getEntrypointSpecs() {
                return defaultCallbacks;
        }

        private static SourceSpec[] sourceSpecs = {

                new CallRetSourceSpec(rslvQuery, new int[] {}),

                new CallRetSourceSpec(new MethodNamePattern(lm, "getProviders"),
                                null),
                new CallRetSourceSpec(new MethodNamePattern(lm, "getProvider"),
                                null),
                new CallRetSourceSpec(new MethodNamePattern(lm,
                                "getLastKnownLocation"), null),
                new CallRetSourceSpec(
                                new MethodNamePattern(lm, "isProviderEnabled"),
null),
                new CallRetSourceSpec(new MethodNamePattern(lm,
"getBestProvider"),
                                null),
                new CallRetSourceSpec(new MethodNamePattern(tm,
                                "getNeighboringCellInfo"), null),
                new CallRetSourceSpec(new MethodNamePattern(tm,
"getCellLocation"),
                                null),

        };

        public SourceSpec[] getSourceSpecs() {
                return sourceSpecs;
        }

        public SourceSpec[] getTransientSourceSpecs() {
                return transientSourceSpecs;
        }

        /**
         * TODO: document!
         */
        private static SinkSpec[] sinkSpecs = {
                new CallArgSinkSpec(actSetResult, new int[] { 2 }),

                new CallArgSinkSpec(rslvQuery, new int[] { 2, 3, 4, 5 }),
                new CallArgSinkSpec(rslvInsert, new int[] { 2 }),
                new CallArgSinkSpec(rslvUpdate, new int[] { 2, 3, 4 }),
```

```
                new CallArgSinkSpec(rslvDelete, new int[] { 2 }),

                new CallArgSinkSpec(actStartActivityForResult, new int[] { 1 }),
                new CallArgSinkSpec(actStartActivityIfNeeded, new int[] { 1 }),
                new CallArgSinkSpec(actStartNextMatchingActivity, new int[] { 1
}),
                new CallArgSinkSpec(actStartActivityFromChild, new int[] { 2 }),

                new CallArgSinkSpec(
                                new MethodNamePattern(smsGsm, "sendTextMessage"),
null),
                new CallArgSinkSpec(
                                new MethodNamePattern(smsGsm, "sendDataMessage"),
null),
                new CallArgSinkSpec(new MethodNamePattern(smsGsm,
                                "sendMultipartTextMessage"), null),

                new CallArgSinkSpec(httpExecute, new int[] {}),

                new CallArgSinkSpec(httpURLGetOutputStream, new int[] {}),
                new CallArgSinkSpec(httpURLSetRequestProperty, new int[] { 1, 2
}),
                new CallArgSinkSpec(cookieSetCookie, new int[] { 1, 2 }),

        };

        private static SinkSpec[] transientSinkSpecs = {
                new CallArgSinkSpec(putBooleanShared, new int[] { 2 }),
                new CallArgSinkSpec(putFloatShared, new int[] { 2 }),
                new CallArgSinkSpec(putIntShared, new int[] { 2 }),
                new CallArgSinkSpec(putLongShared, new int[] { 2 }),
                new CallArgSinkSpec(putStringSetShared, new int[] { 2 }),
                new CallArgSinkSpec(insertDatabase, new int[] { 3 }),
                new CallArgSinkSpec(insertOrThrowDatabase, new int[] { 3 }),
                new CallArgSinkSpec(insertWithOnConflictDatabase, new int[] { 3
}),
                new CallArgSinkSpec(replaceDatabase, new int[] { 3 }),
                new CallArgSinkSpec(replaceOrThrowDatabase, new int[] { 3 }),
                new CallArgSinkSpec(updateDatabase, new int[] { 2 }),
                new CallArgSinkSpec(updateWithOnConflictDatabase, new int[] { 2
}) };

        private static SourceSpec[] transientSourceSpecs = {
                new CallRetSourceSpec(getBooleanShared, null),
                new CallRetSourceSpec(getFloatShared, null),
                new CallRetSourceSpec(getIntShared, null),
                new CallRetSourceSpec(getLongShared, null),
                new CallRetSourceSpec(getStringSetShared, null),
                new CallRetSourceSpec(queryDatabase, null),
                new CallRetSourceSpec(queryWithFactoryDatabase, null),
                new CallRetSourceSpec(rawQueryDatabase, null),
                new CallRetSourceSpec(rawQueryWithFactoryDatabase, null) };

        public SinkSpec[] getSinkSpecs() {
```

```
		return sinkSpecs;
	}

	public SinkSpec[] getTransientSinkSpecs() {
		return transientSinkSpecs;
	}

	private static MethodNamePattern[] callBacks = new MethodNamePattern[] {};

	public static void addPossibleListeners(ClassHierarchy cha) {
		Set<String> ignoreMethods = new HashSet<String>();
		ignoreMethods.add("<init>");
		ignoreMethods.add("<clinit>");
		ignoreMethods.add("registerNatives");
		ignoreMethods.add("getClass");
		ignoreMethods.add("hashCode");
		ignoreMethods.add("equals");
		ignoreMethods.add("clone");
		ignoreMethods.add("toString");
		ignoreMethods.add("notify");
		ignoreMethods.add("notifyAll");
		ignoreMethods.add("finalize");
		ignoreMethods.add("wait");

		List<MethodNamePattern> moreEntryPointSpecs = new
ArrayList<MethodNamePattern>();

		// add default entrypoints from AndroidSpecs.entrypointSpecs
		// Currently adds methods even if they exist in the ignnoreMethods
		// set.
		for (MethodNamePattern mnp : defaultCallbacks) {
			moreEntryPointSpecs.add(mnp);
		}

		for (IClass ic : cha) {
			if (!LoaderUtils.fromLoader(ic,
ClassLoaderReference.Application)) {
				continue;
			}

			// finds all *Listener classes and fetches all methods for the
			// listener
			if (ic.getName().getClassName().toString().endsWith("Listener"))
{
				for (IMethod im : ic.getAllMethods()) {
					// TODO: add isAbstract()?
					if (!ignoreMethods.contains(im.getName().toString())
							&& !im.isPrivate()) {
						moreEntryPointSpecs
		.add(new MethodNamePattern(ic.getName().toString(),
im.getName().toString()));
					}
				}
			}
```

```
                    // not a listener, just find all the methods that start with
                    // "on____"
                    else {
                          for (IMethod im : ic.getAllMethods()) {
                                // TODO: add isAbstract()?
                                if (!ignoreMethods.contains(im.getName().toString())
                                            &&
im.getName().toString().startsWith("on")
                                            && !im.isPrivate()) {
                                      moreEntryPointSpecs
                                                  .add(new
MethodNamePattern(ic.getName()
                                                              .toString(),
im.getName().toString()));
                                }
                          }
                    }
             }

             // entrypointSpecs =
             callBacks = moreEntryPointSpecs
                          .toArray(new
MethodNamePattern[moreEntryPointSpecs.size()]);

      }

      public static MethodNamePattern[] getCallBacks() {
            return callBacks;
      }

}
```

LeakageAnalysis.java

```
package org.distibuteme;

import java.io.IOException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

import org.distibuteme.flow.FlowAnalysis;
import org.distibuteme.flow.InflowAnalysis;
import org.distibuteme.flow.OutflowAnalysis;
import org.distibuteme.util.CGAnalysisContext;
import org.scandroid.domain.CodeElement;
import org.scandroid.domain.DomainElement;
import org.scandroid.domain.IFDSTaintDomain;
import org.scandroid.flow.types.FlowType;
import org.scandroid.spec.ISpecs;
import org.scandroid.util.AndroidAnalysisContext;
import org.scandroid.util.CLISCanDroidOptions;
```

```
import org.scandroid.util.EntryPoints;
import org.scandroid.util.IEntryPointSpecifier;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.google.common.collect.Lists;
import com.ibm.wala.dataflow.IFDS.TabulationResult;
import com.ibm.wala.ipa.callgraph.CGNode;
import com.ibm.wala.ipa.callgraph.Entrypoint;
import com.ibm.wala.ipa.callgraph.propagation.InstanceKey;
import com.ibm.wala.ipa.cfg.BasicBlockInContext;
import com.ibm.wala.ssa.analysis.IExplodedBasicBlock;
import com.ibm.wala.util.MonitorUtil.IProgressMonitor;

public class LeakageAnalysis {
	private static final Logger logger = LoggerFactory
			.getLogger(LeakageAnalysis.class);

	public static void main(String[] args) throws Exception {
		CLISCanDroidOptions options = new CLISCanDroidOptions(args, true);
		logger.info("Loading app.");

		AndroidAnalysisContext analysisContext = new AndroidAnalysisContext(
				options);

		final List<Entrypoint> entrypoints = EntryPoints
				.defaultEntryPoints(analysisContext.getClassHierarchy());

		for (Entrypoint entry : entrypoints) {
			logger.info("Entry point: " + entry);
		}

		if (options.separateEntries()) {
			int i = 1;
			for (final Entrypoint entry : entrypoints) {
				CGAnalysisContext<IExplodedBasicBlock> cgContext = new
CGAnalysisContext<IExplodedBasicBlock>(
						analysisContext, new IEntryPointSpecifier() {
							@Override
							public List<Entrypoint> specify(
									AndroidAnalysisContext
analysisContext) {
								return
Lists.newArrayList(entry);
							}
						});
				logger.info("** Processing entry point " + i + "/"
						+ entrypoints.size() + ": " + entry);
				Map<InstanceKey, String> map = TransientAnalysis
						.runAnalysis(cgContext);
				logger.info("map " + map.size());
				for (Entry<InstanceKey, String> key : map.entrySet()) {
					logger.info("map key= " + key.getKey() + " "
							+ key.getValue());
```

```
                        }

                        analyze(cgContext, null);
                        i++;
                    }
                } else {
                    CGAnalysisContext<IExplodedBasicBlock> cgContext = new
CGAnalysisContext<IExplodedBasicBlock>(
                            analysisContext, new IEntryPointSpecifier() {
                                @Override
                                public List<Entrypoint> specify(
                                        AndroidAnalysisContext
analysisContext) {
                                    return entrypoints;
                                }
                            });
                    Map<InstanceKey, String> map = TransientAnalysis
                            .runAnalysis(cgContext);
                    logger.info("map " + map.size());
                    for (Entry<InstanceKey, String> key : map.entrySet()) {
                        logger.info("map key= " + key.getKey() + " "
                                + key.getValue());
                    }
                    analyze(cgContext, null);
                }
        }

        public static int analyze(
                    CGAnalysisContext<IExplodedBasicBlock> analysisContext,
                    IProgressMonitor monitor) throws IOException {
                try {
                    logger.info("Supergraph size = "
                            + analysisContext.graph.getNumberOfNodes());

                    Map<InstanceKey, String> prefixes;
                    if (analysisContext.getOptions().stringPrefixAnalysis()) {
                        logger.info("Running prefix analysis.");
                        prefixes = TransientAnalysis.runAnalysisHelper(
                                analysisContext.cg, analysisContext.pa);
                        logger.info("Number of prefixes = " +
prefixes.values().size());
                    } else {
                        prefixes = new HashMap<InstanceKey, String>();
                    }

                    ISpecs specs = new AndroidSpecs();

                    logger.info("Running inflow analysis.");
                    Map<BasicBlockInContext<IExplodedBasicBlock>,
Map<FlowType<IExplodedBasicBlock>, Set<CodeElement>>> initialTaints = InflowAnalysis
                            .analyze(analysisContext, prefixes, specs);

                    logger.info("  Initial taint size = " + initialTaints.size());
```

```
                    Logger.info("Running flow analysis.");
                    IFDSTaintDomain<IExplodedBasicBlock> domain = new
IFDSTaintDomain<IExplodedBasicBlock>();
                    TabulationResult<BasicBlockInContext<IExplodedBasicBlock>,
CGNode, DomainElement> flowResult = FlowAnalysis
                                .analyze(analysisContext, initialTaints, domain,
monitor);

                    Logger.info("Running outflow analysis.");
                    Map<FlowType<IExplodedBasicBlock>,
Set<FlowType<IExplodedBasicBlock>>> permissionOutflow = new OutflowAnalysis(
                                analysisContext, specs).analyze(flowResult, domain);
                    Logger.info("  Permission outflow size = "
                                + permissionOutflow.size());

                    Logger.info("");

      Logger.info("================================================================"
);
                    Logger.info("");

                    for (Map.Entry<BasicBlockInContext<IExplodedBasicBlock>,
Map<FlowType<IExplodedBasicBlock>, Set<CodeElement>>> e : initialTaints
                                .entrySet()) {
                          Logger.info(e.getKey().toString());
                          for (Map.Entry<FlowType<IExplodedBasicBlock>,
Set<CodeElement>> e2 : e
                                      .getValue().entrySet()) {
                                Logger.info(e2.getKey() + " <- " + e2.getValue());
                          }
                    }
                    for (Map.Entry<FlowType<IExplodedBasicBlock>,
Set<FlowType<IExplodedBasicBlock>>> e : permissionOutflow
                                .entrySet()) {
                          Logger.info(e.getKey().toString());
                          for (FlowType t : e.getValue()) {
                                Logger.info("    --> " + t);
                          }
                    }

                    return permissionOutflow.size();
              } catch (com.ibm.wala.util.debug.UnimplementedError e) {
                    Logger.error("exception during analysis", e);
              }
              return 0;
      }
}
```

TransientAnalysis.java

```
package org.distibuteme;

import java.util.ArrayList;
import java.util.HashMap;
```

```
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;

import org.scandroid.prefixtransfer.InstanceKeySite;
import org.scandroid.prefixtransfer.PrefixTransferFunctionProvider;
import org.scandroid.prefixtransfer.PrefixVariable;
import org.scandroid.prefixtransfer.TransientTransferGraph;
import org.scandroid.util.CGAnalysisContext;
import org.scandroid.util.EmptyProgressMonitor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


import com.ibm.wala.dataflow.graph.DataflowSolver;
import com.ibm.wala.dataflow.graph.IKilldallFramework;
import com.ibm.wala.dataflow.graph.ITransferFunctionProvider;
import com.ibm.wala.ipa.callgraph.CallGraph;
import com.ibm.wala.ipa.callgraph.propagation.InstanceKey;
import com.ibm.wala.ipa.callgraph.propagation.PointerAnalysis;
import com.ibm.wala.ssa.analysis.IExplodedBasicBlock;
import com.ibm.wala.util.CancelException;
import com.ibm.wala.util.CancelRuntimeException;
import com.ibm.wala.util.graph.Graph;


public class TransientAnalysis {
      private static final Logger logger =
LoggerFactory.getLogger(TransientAnalysis.class);

    public static Map<InstanceKey,String>
runAnalysis(CGAnalysisContext<IExplodedBasicBlock> analysisContext) throws
CancelRuntimeException
    {
        return runAnalysisHelper(analysisContext.cg, analysisContext.pa);
    }

    public static ArrayList<InstanceKey> locateKeys(Map<InstanceKey,String> prefixes,
String s) {
        ArrayList<InstanceKey> keylist = new ArrayList<InstanceKey>();
        for (Entry<InstanceKey,String> e : prefixes.entrySet()) {
            if (e.getValue().contains(s))
                keylist.add(e.getKey());
        }
        return keylist;
    }

    public static Map<InstanceKey,String> runAnalysisHelper(CallGraph cg,
PointerAnalysis pa) throws CancelRuntimeException
    {

        logger.info("*******************************************************");
        logger.info("* Transient Analysis*");
```

```
        final Graph<InstanceKeySite> g = new TransientTransferGraph(pa);
        Logger.info("* The Graph:                                                *");
        Logger.info("************************************************************");
        Iterator<InstanceKeySite> iksI = g.iterator();

        final PrefixTransferFunctionProvider tfp = new
PrefixTransferFunctionProvider();

        IKilldallFramework<InstanceKeySite, PrefixVariable> framework = new
IKilldallFramework<InstanceKeySite, PrefixVariable>()
        {

            public Graph<InstanceKeySite> getFlowGraph() {
                return g;
            }

            public ITransferFunctionProvider<InstanceKeySite, PrefixVariable>
getTransferFunctionProvider() {
                return tfp;
            }

        };

        DataflowSolver<InstanceKeySite, PrefixVariable> dfs = new
DataflowSolver<InstanceKeySite, PrefixVariable>(framework){

            @Override
            protected PrefixVariable makeEdgeVariable(InstanceKeySite src,
                    InstanceKeySite dst) {
                return new PrefixVariable();
            }

            @Override
            protected PrefixVariable makeNodeVariable(InstanceKeySite n,
                    boolean IN) {
                PrefixVariable var = new PrefixVariable();
                return var;
            }

            @Override
            protected PrefixVariable[] makeStmtRHS(int size) {
                return new PrefixVariable[size];
            }

        };

        Logger.info("\n************************************************");
        Logger.info("* Running Analysis");

        try {
            dfs.solve(new EmptyProgressMonitor());
        } catch (CancelException e) {
            throw new CancelRuntimeException(e);
```

```
        }
        Map<InstanceKey,String> keys = new HashMap<InstanceKey,String>();
        iksI = g.iterator();
        while (iksI.hasNext()) {
            InstanceKeySite iks = iksI.next();
            keys.put((InstanceKey)
pa.getInstanceKeyMapping().getMappedObject(iks.instanceID()),
dfs.getOut(iks).knownPrefixes.get(iks.instanceID()));
        }

        return keys;
    }


}
```

TransientContextSelector.java

```
package org.distibuteme.flow;

import com.ibm.wala.classLoader.CallSiteReference;
import com.ibm.wala.classLoader.IMethod;
import com.ibm.wala.ipa.callgraph.AnalysisOptions;
import com.ibm.wala.ipa.callgraph.CGNode;
import com.ibm.wala.ipa.callgraph.Context;
import com.ibm.wala.ipa.callgraph.impl.DefaultContextSelector;
import com.ibm.wala.ipa.callgraph.propagation.InstanceKey;
import com.ibm.wala.ipa.callgraph.propagation.NormalAllocationInNode;
import com.ibm.wala.ipa.callgraph.propagation.ReceiverInstanceContext;
import com.ibm.wala.ipa.callgraph.propagation.cfa.CallerSiteContext;
import com.ibm.wala.ipa.callgraph.propagation.cfa.CallerSiteContextPair;
import com.ibm.wala.ipa.cha.IClassHierarchy;
import com.ibm.wala.types.ClassLoaderReference;
import com.ibm.wala.util.intset.IntSet;

public class TransientContextSelector extends DefaultContextSelector {

    public TransientContextSelector(AnalysisOptions options, IClassHierarchy cha) {
            super(options, cha);
      }

    @Override
    public Context getCalleeTarget(CGNode caller, CallSiteReference site,
            IMethod callee, InstanceKey[] receivers) {

if(callee.getSignature().equals("java.lang.StringBuilder.toString()Ljava/lang/String;
") ||

callee.getSignature().equals("java.lang.StringBuilder.append(Ljava/lang/String;)Ljava
/lang/StringBuilder;") ||

callee.getSignature().equals("java.lang.String.valueOf(Ljava/lang/Object;)Ljava/lang/
String;") ||
```

```
callee.getSignature().equals("java.lang.String.toString()Ljava/lang/String;") ||

callee.getSignature().equals("android.content.SharedPreferences.getString(Ljava/lang/
String;Ljava/lang/String;)Ljava/lang/String;") ||

callee.getSignature().equals("android.content.SharedPreferences.Editor.putString(Ljav
a/lang/String;Ljava/lang/String;)Ljava/lang/String;"))
        {

            if(receivers[0] instanceof NormalAllocationInNode)
            {


if(((NormalAllocationInNode)receivers[0]).getSite().getDeclaredType().getClassLoader(
).equals(ClassLoaderReference.Application)&&!((NormalAllocationInNode)receivers[0]).g
etNode().getMethod().getSignature().contains("android.support")){
                    // create a context based on the site and the receiver
                    return new CallerSiteContextPair(caller,site,new
ReceiverInstanceContext(receivers[0]));
                }
            }
            else
if(callee.getSignature().equals("java.lang.String.valueOf(Ljava/lang/Object;)Ljava/la
ng/String;") ||

callee.getSignature().equals("java.lang.String.toString()Ljava/lang/String;"))
            {
                return new CallerSiteContext(caller,site);
            }
        }
        return super.getCalleeTarget(caller, site, callee, receivers);
    }

    @Override
    public IntSet getRelevantParameters(CGNode node, CallSiteReference call) {
       return super.getRelevantParameters(node,call);
    }
}
```

CGAnalysisContext.java

```
package org.distibuteme.util;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Deque;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
```

```
import org.distibuteme.flow.TransientContextSelector;
import org.scandroid.domain.CodeElement;
import org.scandroid.domain.FieldElement;
import org.scandroid.domain.InstanceKeyElement;
import org.scandroid.util.AndroidAnalysisContext;
import org.scandroid.util.IEntryPointSpecifier;
import org.scandroid.util.ISCanDroidOptions;
import org.scandroid.util.LoaderUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.google.common.collect.Queues;
import com.ibm.wala.classLoader.IClass;
import com.ibm.wala.classLoader.IField;
import com.ibm.wala.classLoader.IMethod;
import com.ibm.wala.dalvik.classLoader.DexIRFactory;
import com.ibm.wala.dataflow.IFDS.ICFGSupergraph;
import com.ibm.wala.dataflow.IFDS.ISupergraph;
import com.ibm.wala.ipa.callgraph.AnalysisCache;
import com.ibm.wala.ipa.callgraph.AnalysisOptions;
import com.ibm.wala.ipa.callgraph.AnalysisScope;
import com.ibm.wala.ipa.callgraph.CGNode;
import com.ibm.wala.ipa.callgraph.CallGraph;
import com.ibm.wala.ipa.callgraph.Entrypoint;
import com.ibm.wala.ipa.callgraph.impl.Everywhere;
import com.ibm.wala.ipa.callgraph.impl.PartialCallGraph;
import com.ibm.wala.ipa.callgraph.propagation.ConcreteTypeKey;
import com.ibm.wala.ipa.callgraph.propagation.InstanceKey;
import com.ibm.wala.ipa.callgraph.propagation.PointerKey;
import com.ibm.wala.ipa.callgraph.propagation.SSAPropagationCallGraphBuilder;
import com.ibm.wala.ipa.cha.ClassHierarchy;
import com.ibm.wala.ssa.IRFactory;
import com.ibm.wala.ssa.ISSABasicBlock;
import com.ibm.wala.ssa.SSACFG;
import com.ibm.wala.ssa.SSACFG.BasicBlock;
import com.ibm.wala.ssa.SSAInstruction;
import com.ibm.wala.types.ClassLoaderReference;
import com.ibm.wala.types.TypeReference;
import com.ibm.wala.util.Predicate;
import com.ibm.wala.util.collections.HashSetFactory;
import com.ibm.wala.util.graph.GraphSlicer;
import com.ibm.wala.util.intset.OrdinalSet;
import com.ibm.wala.util.warnings.Warning;
import com.ibm.wala.util.warnings.Warnings;

public class CGAnalysisContext<E extends ISSABasicBlock> extends
org.scandroid.util.CGAnalysisContext<E>{
      private static final Logger logger =
LoggerFactory.getLogger(CGAnalysisContext.class);

      public CGAnalysisContext(AndroidAnalysisContext analysisContext,
IEntryPointSpecifier specifier)
                  throws IOException {
```

```
		this(analysisContext, specifier, new ArrayList<InputStream>());
	}

	public CGAnalysisContext(AndroidAnalysisContext analysisContext,
IEntryPointSpecifier specifier,
			Collection<InputStream> extraSummaries) throws IOException {
		super(analysisContext, specifier, extraSummaries);
		final AnalysisScope scope = analysisContext.getScope();
		final ClassHierarchy cha = analysisContext.getClassHierarchy();
		final ISCanDroidOptions options = analysisContext.getOptions();

		entrypoints = specifier.specify(analysisContext);
		AnalysisOptions analysisOptions = new AnalysisOptions(scope,
entrypoints);
		for (Entrypoint e : entrypoints) {
			logger.debug("Entrypoint: " + e);
		}
		analysisOptions.setReflectionOptions(options.getReflectionOptions());

		AnalysisCache cache = new AnalysisCache((IRFactory<IMethod>) new
DexIRFactory());

		SSAPropagationCallGraphBuilder cgb;

		if (null != options.getSummariesURI()) {
			extraSummaries.add(new FileInputStream(new
File(options.getSummariesURI())));
		}

		cgb =
AndroidAnalysisContext.makeVanillaZeroOneCFABuilder(analysisOptions, cache,    cha,
scope,
				new TransientContextSelector(analysisOptions, cha), null,
null, null);

		if (analysisContext.getOptions().cgBuilderWarnings()) {
			// CallGraphBuilder construction warnings
			for (Iterator<Warning> wi = Warnings.iterator(); wi.hasNext();) {
				Warning w = wi.next();
				logger.warn(w.getMsg());
			}
		}
		Warnings.clear();

		logger.info("*************************");
		logger.info("* Building Call Graph   *");
		logger.info("*************************");

		boolean graphBuilt = true;
		try {
			cg = cgb.makeCallGraph(cgb.getOptions());
		} catch (Exception e) {
			graphBuilt = false;
			if (!options.testCGBuilder()) {
```

```
                        throw new RuntimeException(e);
                    } else {
                        e.printStackTrace();
                    }
                }

                if (options.testCGBuilder()) {
                    int status = graphBuilt ? 0 : 1;
                    System.exit(status);
                }

                pa = cgb.getPointerAnalysis();
                partialGraph = GraphSlicer.prune(cg, new Predicate<CGNode>() {
                    @Override
                    // CallGraph composed of APK nodes
                    public boolean test(CGNode node) {
                        return LoaderUtils.fromLoader(node,
ClassLoaderReference.Application) || node.getMethod().isSynthetic();
                    }
                });
                if (options.includeLibrary()) {
                    graph = (ISupergraph) ICFGSupergraph.make(cg, cache);
                } else {

                    Collection<CGNode> nodes = HashSetFactory.make();
                    for (Iterator<CGNode> nIter = partialGraph.iterator();
nIter.hasNext();) {
                        nodes.add(nIter.next());
                    }
                    CallGraph pcg = PartialCallGraph.make(cg,
cg.getEntrypointNodes(), nodes);
                    graph = (ISupergraph) ICFGSupergraph.make(pcg, cache);
                }

                oneLevelGraph = GraphSlicer.prune(cg, new Predicate<CGNode>() {
                    @Override
                    public boolean test(CGNode node) {
                        // Node in APK
                        if (LoaderUtils.fromLoader(node,
ClassLoaderReference.Application)) {
                            return true;
                        } else {
                            Iterator<CGNode> n = cg.getPredNodes(node);
                            while (n.hasNext()) {
                                // Primordial node has a successor in APK
                                if (LoaderUtils.fromLoader(n.next(),
ClassLoaderReference.Application))
                                    return true;
                            }
                            n = cg.getSuccNodes(node);
                            while (n.hasNext()) {
                                // Primordial node has a predecessor in APK
                                if (LoaderUtils.fromLoader(n.next(),
ClassLoaderReference.Application))
```

```
                                        return true;
                                }
                                return false;
                        }
                }
        });

        systemToApkGraph = GraphSlicer.prune(cg, new Predicate<CGNode>() {
                @Override
                public boolean test(CGNode node) {

                        if (LoaderUtils.fromLoader(node,
ClassLoaderReference.Primordial)) {
                                Iterator<CGNode> succs = cg.getSuccNodes(node);
                                while (succs.hasNext()) {
                                        CGNode n = succs.next();

                                        if (LoaderUtils.fromLoader(n,
ClassLoaderReference.Application)) {
                                                return true;
                                        }
                                }
                                // Primordial method, with no link to APK code:
                                return false;
                        } else if (LoaderUtils.fromLoader(node,
ClassLoaderReference.Application)) {
                                // see if this is an APK method that was
                                // invoked by a Primordial method:
                                Iterator<CGNode> preds = cg.getPredNodes(node);
                                while (preds.hasNext()) {
                                        CGNode n = preds.next();

                                        if (LoaderUtils.fromLoader(n,
ClassLoaderReference.Primordial)) {
                                                return true;
                                        }
                                }
                                // APK code, no link to Primordial:
                                return false;
                        }

                        // who knows, not interesting:
                        return false;
                }
        });

        if (options.stdoutCG()) {
                for (Iterator<CGNode> nodeI = cg.iterator(); nodeI.hasNext();) {
                        CGNode node = nodeI.next();

                        logger.debug("CGNode: " + node);
                        for (Iterator<CGNode> succI = cg.getSuccNodes(node);
succI.hasNext();) {
```

```
                                Logger.debug("\tSuccCGNode: " +
succI.next().getMethod().getSignature());
                            }
                        }
                }
                for (Iterator<CGNode> nodeI = cg.iterator(); nodeI.hasNext();) {
                        CGNode node = nodeI.next();
                        if (node.getMethod().isSynthetic()) {
                                Logger.trace("Synthetic Method: {}",
node.getMethod().getSignature());
                                Logger.trace("{}",
node.getIR().getControlFlowGraph().toString());
                                SSACFG ssaCFG = node.getIR().getControlFlowGraph();
                                int totalBlocks = ssaCFG.getNumberOfNodes();
                                for (int i = 0; i < totalBlocks; i++) {
                                        Logger.trace("BLOCK #{}", i);
                                        BasicBlock bb = ssaCFG.getBasicBlock(i);

                                        for (SSAInstruction ssaI : bb.getAllInstructions())
{
                                                Logger.trace("\tInstruction: {}", ssaI);
                                        }
                                }
                        }
                }
        }

        public Set<CodeElement> codeElementsForInstanceKey(InstanceKey rootIK) {
                Set<CodeElement> elts = HashSetFactory.make();
                Deque<InstanceKey> iks = Queues.newArrayDeque();
                iks.push(rootIK);

                while (!iks.isEmpty()) {
                        InstanceKey ik = iks.pop();
                        Logger.debug("getting code elements for {}", ik);
                        elts.add(new InstanceKeyElement(ik));
                        final IClass clazz = ik.getConcreteType();
                        final TypeReference typeRef = clazz.getReference();
                        // If an array, recur down into the structure
                        if (typeRef.isArrayType()) {
                                if (typeRef.getArrayElementType().isPrimitiveType()) {
                                        // don't do anything for primitive contents
                                        continue;
                                }
                                OrdinalSet<InstanceKey> pointsToSet =

        pa.getPointsToSet(pa.getHeapModel().getPointerKeyForArrayContents(ik));
                                if (pointsToSet.isEmpty()) {
                                        Logger.debug("pointsToSet empty for array contents,
creating InstanceKey manually");
                                        final IClass contentsClass =
pa.getClassHierarchy().lookupClass(typeRef.getArrayElementType());
                                        if (contentsClass.isInterface()) {
```

```
                                        for (IClass implementor :
analysisContext.concreteClassesForInterface(contentsClass)) {
                                                final InstanceKey contentsIK = new
ConcreteTypeKey(implementor);
                                                final InstanceKeyElement elt = new
InstanceKeyElement(contentsIK);
                                                if (!elts.contains(elt)) {
                                                      elts.add(elt);
                                                      iks.push(contentsIK);
                                                }
                                        }
                                  } else {
                                        InstanceKey contentsIK = new
ConcreteTypeKey(contentsClass);
                                        final InstanceKeyElement elt = new
InstanceKeyElement(contentsIK);
                                        if (!elts.contains(elt)) {
                                              elts.add(elt);
                                              iks.push(contentsIK);
                                        }
                                  }
                            } else {
                                  for (InstanceKey contentsIK : pointsToSet) {
                                        final InstanceKeyElement elt = new
InstanceKeyElement(contentsIK);
                                        if (!elts.contains(elt)) {
                                              elts.add(elt);
                                              iks.push(contentsIK);
                                        }
                                  }
                            }
                            continue;
                      }
                      for (IField field : clazz.getAllInstanceFields()) {
                            Logger.debug("adding elements for field {}", field);
                            final TypeReference fieldTypeRef =
field.getFieldTypeReference();
                            elts.add(new FieldElement(ik, field.getReference()));
                            final IClass fieldClass =
analysisContext.getClassHierarchy().lookupClass(fieldTypeRef);
                            if (fieldTypeRef.isPrimitiveType() || fieldClass == null)
{
                                  continue;
                            } else if (fieldTypeRef.isArrayType()) {
                                  PointerKey pk =
pa.getHeapModel().getPointerKeyForInstanceField(ik, field);
                                  final OrdinalSet<InstanceKey> pointsToSet =
pa.getPointsToSet(pk);
                                  if (pointsToSet.isEmpty()) {
                                        Logger.debug("pointsToSet empty for array
field, creating InstanceKey manually");
                                        InstanceKey fieldIK = new
ConcreteTypeKey(pa.getClassHierarchy().lookupClass(fieldTypeRef));
```

```
                                        final InstanceKeyElement elt = new
InstanceKeyElement(fieldIK);
                                        if (!elts.contains(elt)) {
                                                elts.add(elt);
                                                iks.push(fieldIK);
                                        }
                                } else {
                                        for (InstanceKey fieldIK : pointsToSet) {
                                                final InstanceKeyElement elt = new
InstanceKeyElement(fieldIK);
                                                if (!elts.contains(elt)) {
                                                        elts.add(elt);
                                                        iks.push(fieldIK);
                                                }
                                        }
                                }
                        } else if (fieldTypeRef.isReferenceType()) {
                                PointerKey pk =
pa.getHeapModel().getPointerKeyForInstanceField(ik, field);
                                final OrdinalSet<InstanceKey> pointsToSet =
pa.getPointsToSet(pk);
                                if (pointsToSet.isEmpty() &&
!analysisContext.getClassHierarchy().isInterface(fieldTypeRef)) {
                                        logger.debug("pointsToSet empty for reference
field, creating InstanceKey manually");
                                        InstanceKey fieldIK = new
ConcreteTypeKey(fieldClass);
                                        final InstanceKeyElement elt = new
InstanceKeyElement(fieldIK);
                                        if (!elts.contains(elt)) {
                                                elts.add(elt);
                                                iks.push(fieldIK);
                                        }
                                } else {
                                        for (InstanceKey fieldIK : pointsToSet) {
                                                final InstanceKeyElement elt = new
InstanceKeyElement(fieldIK);
                                                if (!elts.contains(elt)) {
                                                        elts.add(elt);
                                                        iks.push(fieldIK);
                                                }
                                        }
                                }
                        }
                }
        }
        return elts;
    }

    public ISCanDroidOptions getOptions() {
        return analysisContext.getOptions();
    }

    public ClassHierarchy getClassHierarchy() {
```

```
            return analysisContext.getClassHierarchy();
        }

        public AnalysisScope getScope() {
            return analysisContext.getScope();
        }

        public List<Entrypoint> getEntrypoints() {
            return entrypoints;
        }

        public CGNode nodeForMethod(IMethod method) {
            return cg.getNode(method, Everywhere.EVERYWHERE);
        }
}
```