*I. J. Computer Network and Information Security,* **2017, 8, 48-53**
Published Online August 2017 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2017.08.07

# Distributed Malware Detection Algorithm (DMDA)

**Aiman A. Abu Samra, Hasan N. Qunoo, Alaa M. Al Salehi**
Islamic University of Gaza, Gaza city, Palestine
E-mail: aasamra@iugaza.edu.ps, hqunoo@gmail.com, asalehi@iugaza.edu.ps

Received: 18 May 2017; Accepted: 05 July 2017; Published: 08 August 2017

***Abstract*—**The increasing number of malwares has led to an increase in research work on malware analysis studying the malware behavior. The malware tries to leak sensitive information from infected devices. In this paper, we study a specific attack method, which distributes the data source and the point of data loss on different versions of the malware application. That is done using local storage by storing part or all of the vital data to be leaked in the future.

We introduce a Distributed Malware Detection Algorithm (DMDA), which is an algorithm to detect distributed malware on app versions. DMDA proposes a new way to analyze application against redistributed malware. DMDA is created to analyze the data and identify transitional loss points. We test this algorithm on a sample of Android applications published on the Google Play market containing 100 applications, where each application has two versions. The algorithm detected 150 transient data sources, 200 transient loss of data point and two leakages of data. In comparison, this dataset was checked using 56 anti-malware applications but none of them could find any malicious code.

***Index Terms*—**Android, Distributed malware, Malware detection, Transient data sources, Transient sinks.

## I. Introduction

Tremendous increase of android markets and Google's open policy of accepting applications in the Android Marketplace, make it easy for anyone to publish apps and update them. So, the problem of detecting Malwares on mobile devices is an interesting topic. In fact, 86% of detected malwares are old malware repackaged in new apps [1]. However, the antimalware and antivirus tools that we looked at in the scope of this paper, focus only on the current app version. They do not detect an attack distributed over multiple versions of the same application.

There are two types of code analysis that can be used to detect malwares, Static Code Analysis and Dynamic Code Analysis. The difference between two types is that: static program analysis is performed without executing programs, while dynamic analysis is performed by executing programs.

In this paper, we propose a new way to analyze application against redistributed malwares. We also introduce a Distributed Malware Detection Algorithm (DMDA), which is an algorithm to detect distributed malware on application versions.

### *A. 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 the application which is the main method and works on one program architecture, android application has multi-entry points and works on message passing architecture. Android multi-entry points are: Activities, Services, Broadcast Receivers.

### *B. Android Storage Options*

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

Android data storage options are the following: Shared Preferences, Internal Storage, External Storage, SQLite Databases and Network Connection [2].

## II. Related Work

Many works were done in the field of android malware detection. Some works used Static program analysis where the source code is given as input to some automated tool. The tool checks the code without executing it, and yields results. Other works use dynamic analysis where the code is executed. Here we present some of the recent works related to the propagation of malware on the Android platform

In [3] authors explain how to apply clustering techniques in Malware detection of Android applications. Their evaluation is given by clustering two categories of Android applications: business, and tools. They extract the features of the applications from XML-files which contain permissions requested by applications

Enck et al. [4] introduced an approach to convert Dalvik bytecode back to Java bytecode, and then used existing decompilers to obtain the source code of the apps for analysis.

 

Chin et al. [5] showed that apps might be exploitable when servicing external intents. 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 (e.g. the openness can be in the 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 [6]. As demonstrated by [7,8,9], an unprivileged app can access permission-protected resources through privileged apps that do not check permissions. Grace et al. [9] employed an intra-procedural path-sensitive static analysis to discover permission leaks specific to stock apps from multiple device vendors.

In [10] authors introduced a security software that provides comprehensive protection of personal data and mobile telephone from malware and illegal activity of cyber criminals. The developed security software Green Head protects personal smartphones of majority of brands from spam, viruses and unauthorized access.

In [11] authors introduce differential susceptible e-epidemic model for the transmission of malicious codes in a computer network. Authors used Numerical methods to derive the formula for reproduction number (R0) to study the spread of malicious codes in computer network.

AndroidLeaks [12] 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 [13] is a tool for reasoning about data flows in Android applications. It mainly focusses on the inter-component 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.

## III. Research Approach and Tools

In this research, we follow the reverse engineering and the static analysis approach. We also used the Call Graph and Scandroid tools.

### Reverse Engineering

Reverse Engineering is a process of analyzing program code or software 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.

### Static Analysis

Static analyses inspect the program code to derive information about the program's behavior at runtime. As nearly every program has variable ingredients (inputs from a user, files, the internet, etc.) an analysis extract an abstract from concrete program runs. It aims to cover all possibilities by making conservative assumptions. [14]

### 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. Results of the analysis can be used for human understanding of programs, or as a basis for further analysis, such as the analysis that tracks the flow of values between functions. One simple application of call graphs can find functions that are never called.

### WALA

Watson Libraries for Analysis (WALA) is a framework that provides static and dynamic analysis capabilities for Java bytecode and related languages and for JavaScript. [15]

### Scandroid

SCanDroid [13] 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.

## IV. Definitions

To get a precise definition for some terms used in this paper, we define the following terms: Entry point, Source, Sink, Transition Source, Transition Sink, Leak, Transition Leak

*Definition 1 (Entry Point)*

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

*Definition 2 (Sources)*

Sources are calls into resource method returning non-constant value into the application code. This value may be valuable for 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 (Sinks)*

Sinks are 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 Sources)*

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

*Definition 5 (Transient Sinks)*

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

*Definition 6 (Leak)*

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

*Definition 7 (Transient Leak)*

Transient Leak is a call graph path, where the start is a Source resource and the end is a transient Sink resource. Example Application saves contacts data into local storage media.

## V. Distributed over versions Attack Model

As Android is an open environment for development that makes it a target for attacks [7,8,9]. Many other papers talking about malwares attacks, were mentioned in Related Work section. The effort in those papers was focused on a single version of android application. Those papers talked about misused permissions.

However, an attacker can develop an attack based on a distributed multiple version using the accumulated permissions. Android markets including Google Play market support versioning for an android application [16]. Versioning is done through android-mainfest.xml attributes, android: v ersionCode and android: versionName so it is easy for malware producer to distribute his or her malware on multi-version of android application. At the first version, it gets the data from android OS for example contacts and SMSs and store them on its own data. In a later version, the attacker removes the code that was responsible for storing the user data and replaces it with another code that leaks the data to the attacker or to a third party. Figure 1 shows the attack model, which distributes its sources and sinks into two versions of the app

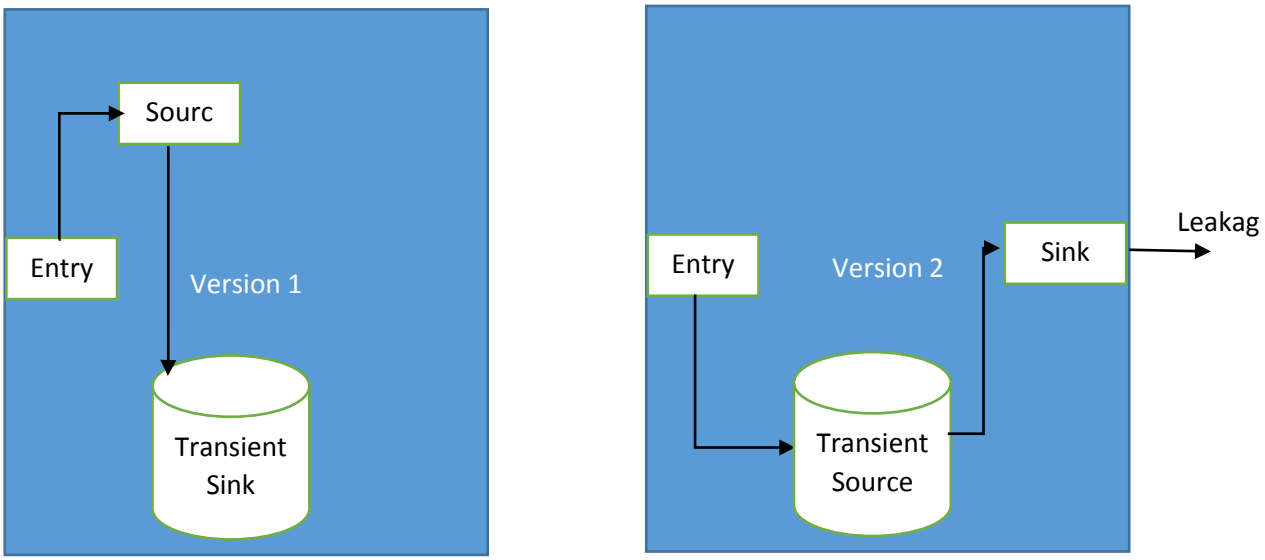


Fig.1. Distributed over versions Attack Model

## VI. DMDA Algorithm

In this section, we explain the proposed DMDA algorithm briefly and then in details. The algorithm implementation and results are explained later in the paper.

### A. *DMDA algorithm main steps:*

1- Build call graph based on entry points
2- Find transient sink and save them with their path to source
3- Replace every transient source with its transient sink path to source
4- Search for data leakage

### B. *DMDA algorithm in details:*

1- Determine entry points of app version
2- Create Call Graph based on S where S is group of E and E is an entry point. Call Graph contains N where N is a group of nodes and D where D is a group of edges
3- Visit Call Graph and determine the following nodes: PI –pure sink, PO –pure source, transient TI –transient sink and TO –transient Source.
4- Use [17] to solve dependencies and reduce reachability.
5- if v == 1 where v is the version of app then

a) call findLeak

 

b) call findTransientSink

6- if transient sink exists then call savetransientSink
7- if v>1

a) call findTransientLeak
b) call findLeak
c) call findTransientSink
d) saveTransientSink

*FindLeak procedure*

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

*SaveTransientSink procedure*

- After finding transient sink check for possible paths used for this sink if there is path to source then save that path
- save the key used to this parameter –example table name for inset statement -

*FindTransientLeak procedure*

- if transient sink saved earlier have the same key of transient sink then it is a transient leak.
  do replace transient sink with transient source paths stored earlier
- else
  ignore this transient source

### *C. Implementation*

We used WALA call graph, which depends on graph reachability concept, and Pointer analysis implementation using kidall`s Framework [18] to follow keys of transient source and transient sinks.

We used many Entry points like:

Android.app.Activity: onCreate, onStart, onRestart, onResume, onPause, onStop, Android.app.Service: onCreate, onStart, onStartCommand, onDestroy, onBind.

ContentProvider: onCreate, query, insert, update, delete. BroadcastReceiver: onReceive.

Then, source and sink of our analysis is determined. In the algorithm Sources have been chosen these like:

- Android.content.ContentResolver. query
- Android.location.LocationManager.[all methods]
- Android.telephony.TelephonyManager.[getNeighboringCellInfo| getCellLocation]

In addition, Sinks were used like:

- 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.
- Android.app.Activity.[starActivity| starActivityForResult| startActivityIfNeeded| startNextMatchingActivity| startActivityFromChild]: these methods can leak data to other applications on the intent send to start their activities.
- Android.content.ContentResolver.[ query|insert| update|delete]: these methods help developers to access Content Provider query, insert, update and delete

These are not all the sources and sinks in android. These are just the ones we used in our work to prove the idea. Some works have studied this topic very well like [4,13,19].

We used also in analysis Transient Sources and Transient Sinks.

Transient sources and sinks are those methods used to store application data into local storage. They cannot be considered as a pure sources and pure sinks because they almost used to store clear data. So, considering them as a source or sink probably results in a false positive malware detection. However, ignoring transient sources and transient sinks may prevent detecting malwares distributed into application versions.

We choose the following methods as transient Sinks:

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. SQ Lite Database[insert| insert Or Throw| insert With On Conflict| replace| replace Or Throw| update| update With On Conflict]: SQ LLite 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 Content Values.

For this paper, 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 be a source of data through their return values.
2- android.database.sqlite.SQLiteDatabase [query| query With Factory| raw Query| raw Query With Factory]: those methods used to retrieve data stored using update, insert and replace methods. These methods could transiently be 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.

Exclusion list

Call graph and static analysis is memory consuming activity. Even simple analysis of an app uses a relatively large memory space. So, WALA uses an exclusion list, which used to exclude unimportant classes from call graph and data flow analysis. We exclude famous libraries and basic java packages. This technique helps WALA to reduce consumed memory and increase productivity.

## VII. Experiments and Analysis

In this section, we explain two experiments made to show and explain the distributed attack and to experiment the effectiveness of DMDA Algorithm. The first experiment focuses on the attack model and how the distribution of a famous malware in two versions makes most of anti-malware unable to detect it. The second experiment checks the effectiveness of DMDA algorithm in finding malware behavior distributed over android app versions and checking these apps.

### A. Attack Model Experiment

In this experiment, we use a combination of DroidKungFu and VirusTotal tools.

DroidKungFu is a Trojan, which although seemingly inoffensive. It can carry out attacks and intrusions: screen logging, stealing personal data, etc.

VirusTotal 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

Of course, a malware distributed over app versions make most of anti-malware unable to detect it. Even for simple, old and famous malwares like DroidKungFu [20]. We use DroidKungFu as an example to explain the attack model. We test DroidKungFu in two versions by VirusTotal, but none of the anti-malwares could catch DroidKungFu.

### B. Effectiveness of DMDA Experiment

For this experiment a group of apps has been chosen including DroidKungFu [20], with two versions of every app these apps taken from [21].

This group contains 100 apps. All of them related to contact's APIs. For every app two versions were chosen. Table 1 shows the results.

Table 1.

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

The algorithm detects over 200 transient sinks and over 150 transient sources, these are not a leakage but they can be developed into a leakage in a future version. The algorithm detects also two leakages. The same dataset was checked using 56 anti-malware applications. All of them failed to detect those positives.

## VIII. Conclusions

Android is the most popular OS for smart phones and it has the biggest number of malwares. In this paper, we propose a new way to analyze mobile apps against redistributed malwares. We also introduce a Distributed Malware Detection Algorithm (DMDA), which is an algorithm to detect distributed malwares over an application versions.

The purpose is to find transient source and transient sink and convert them to their original call graph. This method helps solving malware distribution.

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

To evaluate our algorithm, we tested a group of apps using our algorithm. As a future work plan to enlarge the dataset by adding other existing malwares datasets.

## References

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

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

[3] Aiman A. Abu Samra, Kangbin Yim, Osama A. Ghanem, "Analysis of Clustering Technique in Android Malware Detection" IMIS-2013 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 3-5 July 2013, Asia University, Taichung, Taiwan.

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

[5] 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.

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

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

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

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

[10] 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.
[11] 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.
[12] Syed Arshad and Ashwin Kumar, "Android Application Analysis using Reverse Engineering Techniques and Taint-Aware Slicing". IJCA Proceedings on International Conference on Information and Communication Technologies ICICT(4):5-8, October 2014.
[13] T.J Watson Libraries for Analysis (WALA). http://wala.sf.net
[14] "Versioning Your Applications," Google, [Online]. Available: http://developer.android.com/tools/publishing/versioning.html. [Accessed 2016].
[15] 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.
[16] 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.
[17] "8 Notorious Android Malware Attacks," [Online].Available:http://www.informationweek.com/mobile/8-notorious-android-malware-attacks/d/d-id/1099385.
[18] "F-Droid," [Online]. Available: https://f-droid.org/repository/browse/. [Accessed 2016]
[19] Zhukov Igor, Mikhaylov Dmitry, Starikovskiy Andrey, Dmitry Kuznetsov, Tolstaya Anastasia, Zuykov Alexander. Security Software Green Head for Mobile Devices Providing Comprehensive Protection from Malware and Illegal Activities of Cyber Criminals // Interna- tional Journal of Computer Network and Information Security (IJCNIS). — Vol. 5. — No. 5. — April 2013. —R. 1—8.
[20] Bimal Kumar Mishra, Apeksha Prajapati "Dynamic Model on the Transmission of Malicious Codes in Network", I. J. Computer Network and Information Security, 2013, 10, 17-23
[21] "F-Droid," [Online]. Available: https://f-droid.org/repository/browse/. [Accessed 2016]

## Authors' Profiles

**Dr. Aiman A. Abu Samra**, is an Associate Professor at the Computer Engineering Department at the Islamic University of Gaza. He received his PhD degree from the National Technical University of Ukraine in 1996.

Dr. Aiman was the Assistant Vice President of IT Affairs at IUG between 2011- 2014. He was a supervisor of many Master thesis in mobile computer networks, computer security and other topics. .His research interests include computer networks and mobile computing. Dr. Aiman is a member of the Technical Committee of the International Arab Journal of Information Technology (IAJIT). He was recognized as one of the most active reviewers during the year 2016.

**Hasan Qunoo**, is a young and active lecturer and researcher in computing. Hasan has a PhD and MSc degrees in Computer Security from the University of Birmingham and an extended experience and training in diverse and interactive teaching in the UK and Gaza.

He has taught a number of courses and has been an active member of the curriculum review committees at the department and faculty levels. He is a researcher and lecturer in Computing and Computer Security. He is a member of Computer Security Group at the University of Birmingham. He is now an Assistant Professor at the University of Palestine and was the head of department between 2014 - 2016. He has authored the Palestinian Information Technology Association of Companies strategy (2015-2018). In addition, he has worked in various other technical projects.

**Alaa Al Salehi**, is working in IUG as a team leader for student portal, one of the main systems that serves more than 20,000 students. He runs his own start-up -ZAKI- which is a platform for people whom loves cooking to share their knowledge and experience.

He loves mobile development and adding value to people lives providing solutions for their daily problems. He is recently focusing on android OS, he is the leader and developer of many heterogeneous – web and mobile projects

**How to cite this paper:** Aiman A. Abu Samra, Hasan N. Qunoo, Alaa M. Al Salehi,"Distributed Malware Detection Algorithm (DMDA)", International Journal of Computer Network and Information Security(IJCNIS), Vol.9, No.8, pp.48-53, 2017.DOI: 10.5815/ijcnis.2017.08.07