Twinkle Twinkle Little DroidDream, How I Wonder What You Are?

Fabio Martinelli[‡], Francesco Mercaldo[‡], Vittoria Nardone[*], Antonella Santone[*]
[*]Department of Engineering, University of Sannio, Benevento, Italy
{vnardone, santone}@unisannio.it
[‡]Institute for Informatics and Telematics, National Research Council of Italy (CNR), Pisa, Italy
{fabio.martinelli, francesco.mercaldo}@iit.cnr.it

***Abstract*—Android is the most diffused environment for embedded systems. Not only mobile devices are Android powered, as matter of fact also in automotive and robotics fields, customized Android versions are currently employed. While Android offers several pro, from the stable kernel to the no usage of royalties, there are also cons, one of the most serious is related to the security of the operating systems. In particular, the official market has shown that is not able to block promptly the publication of malicious software. In this paper we discuss a model checking based approach to detect malware related to Android environment. In the evaluation we focus on the DroidDream threat, a malware able to evade the security mechanism provided by the Android official market, obtaining encouraging results.**

I. Introduction

Since the introduction of the open source Android platform for mobile phones by Google, there has been significant interest in the Original Equipment Manufacturer community in order to also make portable i.e., customize Android for other embedded platforms for instance, net-books, set-top boxes and car dashboards. The positive side of making Android available to a lot of platforms would mean that an application at first developed for one device could easily be made portable for another platform with minimal changes needs [1].

As matter of fact, the Android popularity is large and still growing for applications as diverse as automotive [2], wearable computing (wristwatches [3], head-up helmet displays and smart glasses [4]), robotics [5] (for domestic and telepresence applications), multimedia [6] (TVs and media players), vehicle navigation system [7], special-purpose tablets (for instance, the Amazon Kindle) and even near-earth satellites [7].

More recently, a new kind of embedded operating systems is emerging employing Android customized versions [1]. Android may be considered a not usual choice as an embedded OS, but we highlight that Android is an embedded OS, it derives from embedded linux [1]. Android becomes appropriate as an embedded OS outside of tablets and smartphones that require multimedia capabilities and appealing user interface. Android offers several advantages over proprietary embedded operating systems, not the least of which is global familiarity in developing for the platform. Other advantages include a stable kernel, no royalties or licensing fees, and a vast library of open source code and device drivers. These things combine to make creating an embedded system really accessible to manufacturers and developers.

Embedded Android OS is not without downsides: as a matter of fact as the others embedded operating systems, the code base carries significant processing overhead and a larger memory footprint when compared to proprietary embedded OS's. There is functionality that is either not essential toward embedded systems outside of mobile devices [8]. The positive aspect of an open source operating system like Android however, is represented by the fact that individuals and companies alike are free to strip down, modify, and add to the source code to suit their business needs.

Another point of weakness is represented by the fact that Android suffers from a not really good reputation when it comes to security. In particular, the nature of Google Play and other app stores has resulted in a growing distribution of malware itself and new vectors for it.

In order to stem this increasing trend, research community has proposed in last years several approaches to detect mobile malware in Android environment, using both machine learning techniques [9], [10], [11], [12] and formal methods ones [13], [14], [15], [16], [17].

The widespread attack vector in Android malware landascape consists in the so-called repackaged attack [18]: using this technique the attacker obtains the source code of a legitimate application download from a trusted source (i.e., the Google Play market) in order to (re)distribute them with the injection of the malicious payload.

While mobile malware is typically downloaded from third-party markets (i.e., AppChina[2], AppBrain[3]), the malware able to evade the security mechanism provided by the official Android store (i.e., Google Play) is a minimum percentage: the problem in this case is that also few samples are published in the official store they instantly obtain the visibility by all Android-based devices in the moment of the publication. Malware once published is not easy to identify and remove [19], [20], [21], [10], [22], [23], considering that the number of available apps in the Google Play Store is 1 million in July

[1]http://handycodeworks.com/wp-content/uploads/2011/02/linux_versus_android.pdf

[2]www.appchina.com/

[3]https://www.appbrain.com/

2013 and was recently reached at 2.6 million applications in December 2016[4].

As matter of fact when more than 50 Android malware belonging to the DroidDream family were successfully published to the Google play evading the market security mechanism, Lookout estimated between 30,000 and 120,000 users were hit by the infection[5].

This happened why the malware belonging to DroidDream family is able to camouflage their harmful action. As matter of fact, it is able to gain root access to your Android device after the first run of the application. It would then install a second application, which required special permission to uninstall. After that, an infected phone could present several malicious applications installed and send more of the user data to the DroidDream attackers.

Interestingly, DroidDream was designed to do most of its malicious work between 11 p.m. and 8 a.m. when most people would be sleeping and the phone was less likely to be in use. This made it harder to detect abnormal behavior with the infected device.

As the DroidDream spread of infection on Google Play has demonstrated, their security mechanisms are not able to guarantee that the available applications do not perform harmful actions: for these reasons, considering that Android it is installed not only on smartphone and tablet but also in a plethora on embedded and critical system, in this paper we discuss a formal methods based approach to detect mobile malware. In particular the following study is focused on the identification of the DroidDream threats.

The paper proceeds as follows: Section II discusses the current literature, Section III briefly explains our approach, IV discusses the evaluation, finally, conclusion and future work are drawn in Section V.

II. Related Work

In this section we review the current literature related to malware identification with particular regards to malicious family identification.

Neuhaus et al. [24] crawl the vulnerability reports in the Common Vulnerability and Exposures database by using topic models to find prevalent vulnerability types and new trends semi-automatically. They analyze 39393 unique reports until the end of 2009, with the aim of characterizing many vulnerability trends: SQL injection (PHP), buffer overflows, format strings, cross-site request forgery and so on.

Zhao et al. [25] propose a LDA-based method to analyze the trends of network security which consist of three steps: collect data from web sites, extract topics from the collected data, and makes the curves of trends over time. They select 620 documents sorted by time and extract 10 topic from each document.

Song et al. [9] propose a novel threat degree threshold model of dangerous permissions on malware detection. They experiment the method on real mobile devices, using 83 real mobile devices and achieving a 98.8% pass rate, where the versions of Android range from 2.3 to 5.1.

Formal methods have been applied for studying malware in some recent papers. In [16] the authors introduce the specification language CTPL (Computation Tree Predicate Logic) confirming the malicious behavior of thirteen Windows malware variants using as dataset a set of worms dating from 2002 to 2004.

Song et al. present an approach to model as a PushDown System (PDS) Microsoft Windows XP binary programs [26]. They consider 200 malware variants (generated by NGVCK and VCL32 engines) and 8 benign programs.

The tool PoMMaDe [15] is able to detect 600 real malware, 200 malware generated by two malware generators (NGVCK and VCL32), and proves the reliability of benign programs: a Microsoft Windows binary program is modeled as a PDS which allows to track the stack of the program.

Song et al. model mobile applications using a PDS in order to discovery private data leaking working at Smali code level [13].

Jacob and colleagues provide a basis for a malware model, using the Join-Calculus: they build the model [27] considering the system call sequences.

Recently, model checking based approaches have been investigated in [17], [28], [14], [29] to identify the malicious payload in Android malware. Starting from payload behavior definition they formulate logic rules and then test them by using a real world dataset composed by Ransomware, DroidKungFu, Opfake families and update attack samples.

As it emerges from the literature in the last years, formal methods have been applied to detect mobile malware, but at the best knowledge of the authors they have never been applied for identifying specifically the repackaged attack provided by DroidDream family on Android malware.

III. The Method

The approach we propose, as depicted in Figure 1, is the following:

Step 1: Formal specification of DroidDream apps and malware behavior. From manual inspection of the apps and starting form technical reports that present in an informal manner the malware behavior, we extract the key elements, and builds the formulae based on a formalism that is typically a mathematics-based description, using mathematical logic. In this paper we use the mu-calculus logic [30]. Those formulae can be supported by standard model checking tools. Then, also the apps are modeled in with a specification language that can be formally verified using a model checking tool. We use the Calculus of Communicating Systems (CCS) of Milner [31], one one of the most well known formal specification language. CCS contains operators to construct finite processes, parallel processes and some notion of recursion to represent infinite behaviour.

[4]https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[5]http://www.itpro.co.uk/633976/android-droiddream-nightmare-continues

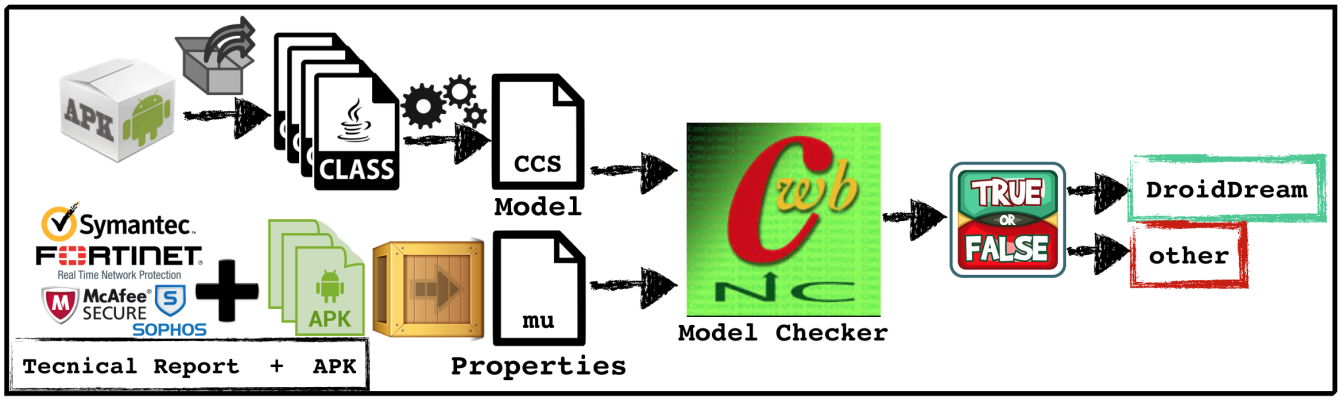

Fig. 1: The Proposed Approach.

Step 2: Model. In this step the apps are abstracted in a way that preserves their security properties, allowing the performance of deep information flow analysis. In our approach, we have chosen the labeled transition system (i.e, automaton) as the representation model, for its ability to represent the control dependency. From the CCS specification defined in the first step, we generated the automaton in an automatic way, starting form a CCS specification and using an operational semantics, For this, we use the Concurrency Workbench of New Century (CWB-NC) [32], one of the most popular environments for verifying concurrent systems.

Step 3: Verification. This step aims at automatically discover DroidDream malware families. The automaton obtained in the second step is used to prove the properties defined in the fist step: using model checking we determine the detection of the DriodDream malware family. In fact, once a model is built, it can be used for verification by model checking. In the model checking approach [33], the system to be verified is modeled as a finite automaton and the property is expressed as a formula of temporal logics. More specifically, this technique relies on translating the system to be verified to formal models that are precise in meaning and amenable to formal analysis, in particular accepted by model checking algorithms. Verifying a property against a model consists of comparing the behavior specified in the property with that permitted by the model. Thus, if the automaton representing an Android application satisfies the formula, we conclude that the app belongs to the DroidDream family, otherwise it can be a trusted app or an application belonging to another malware family. Application of efficient model checking techniques [34] have been used in connection with several disciplines such as biology [35], [36], design patterns [37], information flow [38], clone detection [39] among others. In this paper we address the problem of malware detection exploiting the power of formal methods.

IV. Experiment

In this section we discuss the experiment we performed to evaluate the effectiveness of our approach in recognizing DroidDream payload, discriminating samples belonging to other Android malware families.

A. Dataset

The real world samples examined in the experiment were gathered from the Drebin project's dataset [11], [40]: a very well known collection of malware used in many scientific works, which includes the most diffused Android families.

Malware dataset is also partitioned according to the *malware family*: each family contains samples which have in common several characteristics, like payload installation, the kind of attack and events that trigger malicious payload [18].

Table I shows the 10 malware families with the biggest number of applications in our malware dataset with installation type, kind of attack and event activating malicious payload.

The malware was retrieved from the Drebin project [11], [40] taking into account the top 10 most populous families.

B. Evaluation

To estimate the detection performance of our methodology we compute the metrics of precision and recall, F-measure (Fm) and Accuracy (Acc), defined as follows:

$$PR = \frac{TP}{TP + FP}; \quad RC = \frac{TP}{TP + FN};$$

$$Fm = \frac{2PR\ RC}{PR + RC}; \quad Acc = \frac{TP + TN}{TP + FN + FP + TN}$$

where TP is the number of malware that was correctly identified in the DroidDream family (True Positives), TN is the number of malware correctly identified as not belonging to the DroidDream family (True Negatives), FP is the number of malware that was incorrectly identified in the DroidDream family (False Positives), and FN is the number of malware that was not identified as belonging to the DroidDream family (False Negatives).

Table II shows the results obtained using our method.

TABLE I: Families in Drebin dataset with details of the installation method (standalone, repackaging, update), the kind of attack (trojan, botnet), the events that trigger the malicious payload and a brief family description.

Family	Installation	Attack	Activation	Description
FakeInstaller	s	t,b		server-side polymorphic family
Plankton	s,u	t,b		it uses class loading to forward details
DroidKungFu	r	t	boot,batt,sys	it installs a backdoor
GinMaster	r	t	boot	malicious service to root devices
BaseBridge	r,u	t	boot,sms,net,batt	it sends information to a remote server
Adrd	r	t	net,call	it compromises personal data
Kmin	s	t	boot	it sends info to premium-rate numbers
Geinimi	r	t	boot,sms	first Android botnet
DroidDream	r	b	main	botnet, it gained root access
Opfake	r	t		first Android polymorphic malware

TABLE II: Performance Evaluation

DroidDream	Malware	TP	FP	FN	TN	PR	RC	Fm	Acc
81	780	77	66	4	714	0.54	0.95	**0.69**	**0.92**

We consider in the column *DroidDream* the samples belonging to *DroidDream* family, while in the column *Malware* the malware samples belonging to others families considered in the study: the detail about the malicious payload of the family we considered is shown in Table I. We demonstrate the effectiveness of our approach evaluating 81 malware belonging to *DroidDream* family and 780 malware belonging to other families.

Results in Table II seems to be very promising: we obtain an Accuracy equal to 0.92. Concerning the DroidDream results, we are not able to identify the malicious payloads of just 4 samples on 81. It is worth noting the above values are also due to the fact that the dataset is unbalanced, i.e., 81 malware belonging to *DroidDream* family and 780 belonging to other real-world widespread Android malware families.

V. Conclusion and Future Work

The Android operation system is the most widespread: it is customized from device mobile vendors, but also from robotics and automotive companies. For this reason aspects related to its security are becoming extremely important. In order to limit the diffusion of malicious software targeting this platform, in this paper we proposed a model checking based approach to detect Android mobile malware. We evaluate our method using real-world malware, focusing the study on the identification of the DroidDream malware, one of the most diffused threats employing repackaged attacks to perform the harmful actions.

We plan to extend this work analyzing mobile malware belonging to other platforms (i.e., Apple iOS) in order to demonstrate the portability of the proposed approach and to investigate the use of equivalence checking [41], [42].

Acknowledgements

This work has been partially supported by H2020 EU-funded projects NeCS and C3ISP and EIT-Digital Project HII and PRIN "Governing Adaptive and Unplanned Systems of Systems"

References

[1] K. Yaghmour, *Embedded Android: Porting, Extending, and Customizing*. " O'Reilly Media, Inc.", 2013.

[2] J. Cai, J. Wu, M. Wu, and M. Huo, "A bluetooth toy car control realization by android equipment," in *Transportation, Mechanical, and Electrical Engineering (TMEE), 2011 International Conference on*. IEEE, 2011, pp. 2429–2432.

[3] Q. Do, B. Martini, and K.-K. R. Choo, "Is the data on your wearable device secure? an android wear smartwatch case study," *Software: Practice and Experience*, 2016.

[4] E. Ackerman, "Google gets in your face [2013 tech to watch]," *IEEE Spectrum*, vol. 50, no. 1, pp. 26–29, 2013.

[5] J.-H. Oh, D. Hanson, W.-S. Kim, Y. Han, J.-Y. Kim, and I.-W. Park, "Design of android type humanoid robot albert hubo," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 1428–1433.

[6] N. Kuzmanovic, T. Maruna, M. Savic, G. Miljkovic, and D. Isailovic, "Google's android as an application environment for dtv decoder system," in *Consumer Electronics (ISCE), 2010 IEEE 14th International Symposium on*. IEEE, 2010, pp. 1–5.

[7] L.-P. Nong, L.-H. Wang, and Y.-P. Huang, "Application research of android in embedded vehicle navigation system," *Computer Engineering and Design*, vol. 31, no. 11, pp. 2473–2476, 2010.

[8] C. Maia, L. M. Nogueira, and L. M. Pinho, "Evaluating android os for embedded real-time systems," in *6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, 2010, pp. 63–70.

[9] J. Song, C. Han, K. Wang, J. Zhao, R. Ranjan, and L. Wang, "An integrated static detection and analysis framework for android," *Pervasive and Mobile Computing*, 2016.

[10] G. Canfora, F. Mercaldo, and C. A. Visaggio, "An hmm and structural entropy based detector for android malware: An empirical study," *Computers & Security*, vol. 61, pp. 1–18, 2016.

[11] D. Arp, M. Spreitzenbarth, M. Huebner, H. Gascon, and K. Rieck, "Drebin: Efficient and explainable detection of android malware in your pocket," in *Proceedings of 21th Annual Network and Distributed System Security Symposium (NDSS)*. IEEE, 2014.

[12] G. Canfora, F. Mercaldo, and C. A. Visaggio, "A classifier of malicious android applications," in *Proceedings of the 2nd International Workshop on Security of Mobile Applications, in conjunction with the International Conference on Availability, Reliability and Security*. IEEE, 2013.

[13] F. Song and T. Touili, "Model-checking for android malware detection." Springer, 2014.

[14] F. Mercaldo, V. Nardone, A. Santone, and C. A. Visaggio, "Ransomware steals your phone. formal methods rescue it," in *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*. Springer, 2016, pp. 212–221.

[15] F. Song and T. Touili, "Pommade: Pushdown model-checking for malware detection," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013.

[16] J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith, "Detecting malicious code by model checking." Springer, 2005.

[17] P. Battista, F. Mercaldo, V. Nardone, A. Santone, and C. A. Visaggio, "Identification of android malware families with model checking," in *International Conference on Information Systems Security and Privacy*. SCITEPRESS, 2016.

[18] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 95–109.

[19] F. Mercaldo, C. A. Visaggio, G. Canfora, and A. Cimitile, "Mobile malware detection in the real world," in *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 744–746.

[20] G. Canfora, F. Mercaldo, and C. A. Visaggio, "A classifier of malicious android applications," in *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*. IEEE, 2013, pp. 607–614.

[21] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Acquiring and analyzing app metrics for effective mobile malware detection," in *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*. ACM, 2016, pp. 50–57.

[22] ——, "Detecting android malware using sequences of system calls," in *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*. ACM, 2015, pp. 13–20.

[23] G. Canfora, F. Mercaldo, and C. A. Visaggio, "Mobile malware detection using op-code frequency histograms," in *Proceedings of International Conference on Security and Cryptography (SECRYPT)*, 2015.

[24] S. Neuhaus and T. Zimmermann, "Security trend analysis with cve topic models," in *Software reliability engineering (ISSRE), 2010 IEEE 21st international symposium on*. IEEE, 2010, pp. 111–120.

[25] Y.-B. Zhao, S.-M. Liu, and S.-Q. Guo, "Extraction and prediction of hot topics in network security," in *Computer Science and Network Security, 2014 International Conference on*, 2014, pp. 347–353.

[26] F. Song and T. Touili, "Efficient malware detection using model-checking." Springer, 2001.

[27] G. Jacob, E. Filiol, and H. Debar, "Formalization of viruses and malware through process algebras," in *International Conference on Availability, Reliability and Security (ARES 2010)*. IEEE, 2010.

[28] F. Mercaldo, V. Nardone, A. Santone, and C. A. Visaggio, "Download malware? No, thanks. How formal methods can block update attacks," in *Formal Methods in Software Engineering (FormaliSE), 2016 IEEE/ACM 4th FME Workshop on*. IEEE, 2016.

[29] ——, "Hey malware, i can find you!" in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2016 IEEE 25th International Conference on*. IEEE, 2016, pp. 261–262.

[30] C. Stirling, "An introduction to modal and temporal logics for ccs," in *Concurrency: Theory, Language, And Architecture*, 1989, pp. 2–20.

[31] R. Milner, *Communication and concurrency*, ser. PHI Series in computer science. Prentice Hall, 1989.

[32] R. Cleaveland and S. Sims, "The ncsu concurrency workbench," in *CAV*. Springer, 1996.

[33] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT Press, 2001.

[34] R. Barbuti, N. De Francesco, A. Santone, and G. Vaglini, "Reduced models for efficient CCS verification," *Formal Methods in System Design*, vol. 26, no. 3, pp. 319–350, 2005.

[35] M. Ceccarelli, L. Cerulo, G. D. Ruvo, V. Nardone, and A. Santone, "Infer gene regulatory networks from time series data with probabilistic model checking," in *3rd IEEE/ACM FME Workshop on Formal Methods in Software Engineering, FormaliSE 2015, Florence, Italy, May 18, 2015*. IEEE Computer Society, 2015, pp. 26–32.

[36] M. Ceccarelli, L. Cerulo, and A. Santone, "De novo reconstruction of gene regulatory networks from time series data, an approach based on formal methods," *Methods*, vol. 69, no. 3, pp. 298 – 305, 2014.

[37] M. Bernardi, M. Cimitile, G. De Ruvo, G. Di Lucca, and A. Santone, "Integrating model driven and model checking to mine design patterns," *Communications in Computer and Information Science*, vol. 586, pp. 99–117, 2016.

[38] R. Barbuti, N. De Francesco, A. Santone, and L. Tesei, "A notion of non-interference for timed automata," *Fundam. Inform.*, vol. 51, no. 1-2, pp. 1–11, 2002. [Online]. Available: http://content.iospress.com/articles/fundamenta-informaticae/fi51-1-2-02

[39] A. Santone, "Clone detection through process algebras and java bytecode," in *Proceeding of the 5th ICSE International Workshop on Software Clones, IWSC 2011, Waikiki, Honolulu, HI, USA, May 23, 2011*. ACM, 2011, pp. 73–74.

[40] M. Spreitzenbarth, F. Echtler, T. Schreck, F. C. Freling, and J. Hoffmann, "Mobilesandbox: Looking deeper into android applications," in *28th International ACM Symposium on Applied Computing (SAC)*. ACM, 2013.

[41] N. De Francesco, G. Lettieri, A. Santone, and G. Vaglini, "Heuristic search for equivalence checking," *Software and System Modeling*, vol. 15, no. 2, pp. 513–530, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10270-014-0416-2

[42] ——, "Grease: A tool for efficient "nonequivalence" checking," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 3, pp. 24:1–24:26, 2014.