Ransomware Steals Your Phone. Formal Methods Rescue It

Francesco Mercaldo, Vittoria Nardone, Antonella Santone, Corrado Visaggio

Ransomware Steals your Phone. Formal Methods Rescue it.

Francesco Mercaldo, Vittoria Nardone, Antonella Santone, Corrado Aaron Visaggio
{fmercaldo, vnardone, santone, visaggio}@unisannio.it

[1]Department of Engineering, University of Sannio, Italy

Abstract. Ransomware is a recent type of malware which makes inaccessible the files or the device of the victim. The only way to unlock the infected device or to have the keys for decrypting the files is to pay a ransom to the attacker. Commercial solutions for removing ransomware and restoring the infected devices and files are ineffective, since this malware uses a very robust form of asymmetric cryptography and erases shadow copies and recovery points of the operating system. Literature does not count many solutions for effectively detecting and blocking ransomware and, at the best knowledge of the authors, formal methods were never applied to identify ransomware. In this paper we propose a methodology based on formal methods that is able to detect the ransomware and to identify in the malware's code the instructions that implement the characteristic instructions of the ransomware. The results of the experimentation are strongly encouraging and suggest that the proposed methodology could be the right way to follow for developing commercial solutions that could successful intercept the ransomware and blocking the infections it provokes.

Keywords: Malware; Android; Security; Formal Methods; Temporal Logic.

1 Introduction and Motivation

Ransomware is a recent kind of malware that spread out mainly in last couple of years, and it is particularly aggressive for two reasons: on one hand it uses very effective mechanisms of infection based mainly on techniques of social engineering like sophisticated phishing (by mail or chat), and on the other hand it makes completely inaccessible the data on the infected machine, as it cyphers all the files with a strong asymmetric key cryptographic algorithm.

The ransomware is still increasing its capability to harm the victim's device, prevent the restore of the data or device, and evade detection. As a matter of fact, the more recent releases of this malware are able to recognize when they are executed in a virtual environment, which is often used for creating a sandbox where safely executing a program for studying its behavior and understanding if it launches a malicious payload or not. Additionally, recent ransomware is

equipped with anti-debugging techniques, which is another way to evade detection as hindering the scanning of anti-malware.

Statistics from US governative agencies show that Cryptolocker infected in 2014: 336,856 machines in USA, 4,593 in UK, 25,841 in Canada, 15,427 in Australia, 1,832 in India, 100,448 in other countries. At its peak, CryptoLocker, a kind of ransomware, was infecting around 50,000 computers per month. According to SCMagazine[1] the CryptoWall, another ransomware, in a roughly five-month period infected 625,000 victims worldwide, encrypting 5.25 billion files, collecting more than $1.1 million in ransoms. The malware tries to delete shadow copies of the system through vssadmin.exe, so that the victim cannot return to previous system restore points too.

Moreover ransomware is invading the smartphone world: Kaspersky labs found 1,113 new ransomware samples targeting Android devices in the first quarter of 2015, which is a 65% increase in the number of mobile ransomware samples with respect to those collected in 2014[2]. This is a dangerous trend since ransomware is designed to extort money, damage personal data, and block infected devices. Once the device is infected, the attacker asks the victim to pay a ransom in order to obtain the key for decrypting the files or restoring the control of the smartphone.

As the evidence of the high infections rate demonstrates, commercial anti-malware solutions are mainly ineffective to detect ransomware.

For this reason we propose a technique for specifically detecting ransomware on smartphone devices that is completely based on formal methods. The technique has been proved to be very effective as the evaluation produced an F-measure of detection equal to 0.99 on a dataset of $2,477$ samples. Additionally, the technique is able to localize in the code the peculiar instructions that implement the stages of infection, and the activation of the payload, which provides fundamental pieces of information to build both effective detectors and removal systems for ransomware. Moreover, at the best knowledge of the authors, literature counts only two works that propose a method to detect mobile ransomware [2, 20] and that are compared with ours in the section of related work.

The paper proceeds as follows: Section 2 describes and motivates our detection method; Section 3 illustrates the results of experiments; Section 4 discusses the related work; finally, conclusions are drawn in Section 5.

2 The Methodology

In this section we present our methodology for the detection of Android ransomware malware using model checking. While model checking was originally developed to verify the correctness of systems against specifications, recently it

[1] http://www.scmagazine.com/cryptowall-surpasses-cryptolocker-in-infection-rates/article/368920/

[2] https://securelist.com/analysis/quarterly-malware-reports/69872/it-threat-evolution-in-q1-2015/

has been highlighted in connection with a variety of disciplines see [1,7] Moreover, great advancements have been made to tackle the limitation of this technique due to its high time and memory requirements, see [4, 9–11]. In this paper we present the use of model checking in the security field.

2.1 Formal Methods for Ransomware Detection

The approach is structured in three main sub-processes.

Formal Model Construction. This first sub-process aims at deriving formal models starting from the Java Bytecode. The Bytecode of the app under study is parsed and suitable formal models of the system are produced. More precisely, the bytecode of the analysed app that resides in a class folder or in JAR files is fed to a custom parser, based on the Apache Commons Bytecode Engineering Library (BCEL)[3]. The parsed Java Bytecode of the .class files are successively translated into formal models. In our approach Calculus of Communicating Systems (CCS) [15] has been exploited. CCS [15] is one of the most well known process algebras. A Java Bytecode-to-CCS transforming function has been defined for each instruction of the Java Bytecode. We associate a new CCS process to each Java Bytecode instruction. This translation has to be performed only one time for each app to be analysed and it has been completely automated. Each Java Bytecode instruction that is not a (conditional or unconditional) jump is represented by a process that, using the operator (“.”), invokes the process corresponding to its successive instruction. Conditional jumps are instead specified as non-deterministic choices. An unconditional jump is represented by a CCS process that invokes the corresponding process of the jump target.

Temporal Logic Properties Construction. This second sub-process aims to define the characteristic behaviour of a ransomware by means of the construction of the temporal logic properties. This step tries to recognize specific and distinctive features of the ransomware behaviour with respect to all the other malware families and to goodware too. Thus, this specific behaviour is written as a set of properties. To specify the properties, we manually inspected a few samples in order to find the ransomware malicious behavior implementation at Bytecode level. In our approach, the mu-calculus logic [19] is used, which is a branching temporal logic to express behavioural properties.

Ransomware Family Detection. Finally, a formal verification environment, including a model checker, is invoked to recognise the ransomware family. This sub-process checks the sets of logic properties obtained from the ransomware malware family characterization against the CCS model of the app. In our approach, we invoke the Concurrency Workbench of New Century (CWB-NC) [8] as formal verification environment. When the result of the CWB-NC model checker is *true*, it means that the app under analysis belongs to the ransomware family, *false* otherwise. Thanks to very detailed CCS model and the logic formulae we are able to reach a good accuracy of the overall results, as explained in the following section.

[3] http://commons.apache.org/bcel/

To the Authors' knowledge, model checking has never used before for the ransomware detection. The main distinctive features of the approach proposed in this paper are the use of formal methods, the identification of the ransomware through the Java Bytecode and the definition of a fully static approach. More precisely, our methodology exploits the Bytecode representation of the analysed apps. Detecting Android ransomware through the Bytecode and not directly on the source code has several benefits: (i) independence of the source programming language; (ii) recognition of malware families without decompilation even when source code is lacking; (iii) easiness of parsing a lower-level code; (iv) independence from obfuscation.

Another important feature of our approach is that we try to reuse existing model checkers avoiding the design of custom-made model checker. In fact our goal is to recognise ransomware with the criteria of reusing existing checking technologies. Model checkers, especially the most widely used ones, are extremely sophisticated programs that have been crafted over many years by experts in the specific techniques employed by the tool. A re-implementation of the algorithms in these tools could likely yield worst performance.

3 Results and Discussion

3.1 Empirical Evaluation Procedure

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

3.2 Experimental Dataset

The real world samples examined in the experiment were gathered from three different datasets. The first one is a collection of freely available 672 samples[4] and 11[5] Android ransomware samples. The samples are labelled as ransomware, koler, locker, fbilocker and scarepackage [2] and appeared from December 2014

[4] http://ransom.mobi/
[5] http://contagiominidump.blogspot.it/

Table 1: Dataset used in the Experiment

Dataset	Original Samples	Morphed Samples	#Samples for Category
Ransomware	683	594	1,277
Other Malware	600	0	600
Trusted	600	0	600
Total	1,883	594	**2,477**

to June 2015. The second one is the Drebin project's dataset [3,18], a very well known collection of malware used in many scientific works, which includes the most diffused Android families.

Each malware sample in these datasets is labelled according to the *malware family* it belongs to: each family comprehends samples which have in common the same payload. This collection does not contain ransomware samples: we use this dataset to check the true positives.

The last one is a dataset of trusted applications crawled from Google Play[6], by using a script which queries a python API[7] to search and download apps. The downloaded applications belong to all the 26 different available categories (i.e., Books & Reference, Lifestyle, Business, Live Wallpaper, Comics, Media & Video, Communication, Medical, Education, Music & Audio, Finance & News, Magazines, Games, Personalization, Health & Fitness, Photography, Libraries & Demo, Productivity, Shopping, Social, Sport, Tools, Travel, Local & Transportation, Weather, Widgets). The applications retrieved were among the most downloaded in their category and were free.

The trusted applications were collected between April 2015 and January 2016 and were later analysed with the VirusTotal service[8], a service able to run 57 different antimalware software (i.e., Symantec, Avast, Kasperky, McAfee, Panda, and others) on the app: the analysis confirmed that the crawled applications did not contain malicious payload. We use this dataset to check the true positives.

Furthermore, we developed a framework[9] able to inject several obfuscation levels in Android applications: (i) changing package name; (ii) identifier renaming; (iii) data encoding; (iv) call indirections; (v) code reordering; (vi) junk code insertion.

These injections were aimed at generating morphed versions of the applications belonging to the ransomware dataset. Previous works [6] demonstrated that antimalware solutions fail to recognize the malware after these transformations. We applied our method to the morphed dataset in order to verify if it loses its effectiveness, too, or it keeps on recognizing the malware also after they have been altered. Table 1 provides the details of the full collection of 2,477 samples used to test the effectiveness of our method. Regarding the ransomware samples, Table 1 shows the number of original and morphed samples we tested; in some cases our framework was not able to disassemble some of the selected

[6] https://play.google.com
[7] https://github.com/egirault/googleplay-api
[8] https://www.virustotal.com/
[9] https://github.com/faber03/AndroidMalwareEvaluatingTools

Table 2: 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

samples, this is the reason why we had to discard them and we consider a lower number of morphed samples if compared with original ones. In order to test the capacity of our rules to identify exclusively ransomware samples, we include in the dataset both trusted and malware samples from other families (respectively *Trusted* and *Other Malware*).

Table 2 provides a brief description of the payload brought by the malware families labelled as *Other Malware*, i.e., malware that is not ransomware.

We test 60 samples for each family. The malware was retrieved from the Drebin project [3, 18] (we take into account the top 10 most populous families).

3.3 Evaluation

As a baseline for evaluating the performances of our solution, we compare the results obtained with our method with those produced by the top 10 ranked mobile antimalware solutions from AVTEST[10], an independent Security Institute which each year provides an analysis of the performances of the antimalware software. We built this baseline, by submitting our original and morphed samples to the VirusTotal API[11], which allows to run the above mentioned antimalware.

Table 3 shows the evaluation between the top 10 antimalware solutions and our method with the original ransomware samples and with the morphed ones.

We consider only the samples and the percentage identified in the right family (column "ident" and the percentage in column "%ident") in Table 3. We also report the samples detected as malicious but not identified in the right family and the samples not recognized as malware (column "unident").

With regards to Table 3 we notice that BitDefender shows better performance in family identification for *Ransomware* original samples. Instead, with regards to morphed samples, antimalware performance decreases dramatically, indeed BitDefender is able to identify only 86 samples. The worst antimalware in *Ransomware* identification is Alibaba, able to correctly classify just 3 original samples and 0 morphed samples.

[10] https://www.av-test.org/en/antivirus/mobile-devices/
[11] https://www.virustotal.com/

Table 3: Top 10 Signature-Based Antimalware Evaluation Against Our Method.

Antimalware	**Original**			**Morphed**		
	%ident.	#ident.	#unident.	%ident.	#ident.	#unident.
AhnLab	*13.76%*	94	589	*5.22%*	31	563
Alibaba	*0.44%*	3	680	*0%*	0	594
Antiy	*13.18%*	90	593	*4.04%*	24	570
Avast	*27.52%*	188	495	*6.4%*	38	556
AVG	*3.22%*	22	661	*1.51%*	9	585
Avira	*19.76%*	135	548	*12.46%*	74	520
Baidu	*14.34%*	98	585	*6.7%*	41	553
BitDefender	*28.26%*	193	490	*14.47%*	86	508
ESET-NOD32	*20.35%*	139	544	*8.58%*	51	543
GData	*27.96%*	191	492	*7.91%*	47	547
Our Method	***99.56%***	**680**	**3**	***99.49%***	**591**	**3**

Table 4: Performance Evaluation

Formula	**# Samples**	**TP**	**FP**	**FN**	**TN**	**PR**	**RC**	**Fm**	**Acc**
Ransomware	2,477	1,271	0	6	1,200	1	0.99	**0.99**	**0.99**

Due to the novelty of the problem, antimalware solutions are not still specialized in family identification; this is the reason why most of antimalware are unskilled to detect families. Another problem is that current antimalware are not able to detect malware when the signature mutates: their performance decreases dramatically with morphed samples. On the contrary, the detection done by our method is barely affected by the code transformations, so it is independent from the signature.

Table 4 shows the results obtained using our method. We consider the sum of original and morphed samples: the detail about the number of original and morphed samples is shown in Table 1.

Results in Table 4 seems to be very promising: we obtain an Accuracy and a F-measure equal to 0.99. Concerning the ransomware results, we are not able to identify the malicious payloads of just 6 samples (i.e., 3 originals and 3 morphed) on $1,277$.

4 Related Work

In this section we review the current literature related to ransomware detection and formal methods applied to the detection of mobile malware.

As we stated previously, literature counts only two works about the detection of Android ransomare.

The first one proposes HelDroid [2]: the approach includes a text classifier based on NLP features, a lightweight Smali emulation technique to detect locking strategies, and the application of taint tracking for detecting file-encrypting

flows. The main weakness of HelDroid is represented by the text classifier: the authors train it on generic threatening phrases, similar to those that typically appear in ransomware or scareware samples. In addiction, like whatever machine learning approach, HelDroid needs to train the classifier in order to label a samples as a ransomware: the detection capability of the model is related to the training dataset.

The other work in literature exploring the ransomare detection in mobile world is [20]. The authors illustrate a possible design of a static and dynamic analysis based solution, without implementing it. Their goal is to build a better performance tool in order to help to understand what should do to approach for a successful detection of Android ransomware.

Formal methods have been applied for studying malware in some recent papers, see [12,13,16,17]. Recently, the possibility to identify the malicious payload in Android malware using a model checking based approach has been explored in [5, 14]. Starting from payload behavior definition they formulate logic rules and then test them by using a real world dataset composed by DroidKungFu, Opfake families and update attack samples. However, 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 ransomware attack on Android malware.

5 Conclusions

Ransomware is a new type of malware that restricts access to the infected smartphone and it demands the user to pay a ransom to the attacker in order to remove the restriction. Ransomware samples are able to encrypt files on the infected device, which become difficult or impossible to decrypt without paying the ransom for the encryption key.

In this paper we propose a technique based on formal methods able to detect ransomware behaviour in Android platform. We obtain encouraging results on a dataset of 2,477 samples: 1 precision and 0.99 recall, overcoming in terms of effectiveness the top 10 popolous commercial antimalware.

As future work we are going to extend our solution using a ransomware dataset for different environment, like Windows Mobile and iOS in order to experiment the portability of our method.

References

1. Anastasi, G., Bartoli, A., Francesco, N.D., Santone, A.: Efficient verification of a multicast protocol for mobile computing. Comput. J. 44(1), 21–30 (2001)
2. Andronio, N., Zanero, S., Maggi, F.: Heldroid: Dissecting and detecting mobile ransomware. In: Research in Attacks, Intrusions, and Defenses, pp. 382–404. Springer (2015)
3. Arp, D., Spreitzenbarth, M., Huebner, M., Gascon, H., Rieck, K.: 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)
4. Barbuti, R., Francesco, N.D., Santone, A., Vaglini, G.: LORETO: A tool for reducing state explosion in verification of LOTOS programs. Softw., Pract. Exper. 29(12), 1123–1147 (1999)
5. Battista, P., Mercaldo, F., Nardone, V., Santone, A., Visaggio, C.A.: Identification of android malware families with model checking. In: International Conference on Information Systems Security and Privacy. SCITEPRESS (2016)
6. Canfora, G., Di Sorbo, A., Mercaldo, F., Visaggio, C.: Obfuscation techniques against signature-based detection: a case study. In: Proceedings of Workshop on Mobile System Technologies. IEEE (2015)
7. Ceccarelli, M., Cerulo, L., Ruvo, G.D., Nardone, V., Santone, A.: 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. pp. 26–32. IEEE (2015)
8. Cleaveland, R., Sims, S.: The ncsu concurrency workbench. In: Alur, R., Henzinger, T.A. (eds.) CAV. Lecture Notes in Computer Science, vol. 1102. Springer (1996)
9. De Francesco, N., Lettieri, G., Santone, A., Vaglini, G.: Heuristic search for equivalence checking. Software & Systems Modeling (2014)
10. Francesca, G., Santone, A., Vaglini, G., Villani, M.L.: Ant colony optimization for deadlock detection in concurrent systems. In: Proceedings of the 35th Annual IEEE International Computer Software and Applications Conference, COMPSAC 2011, Munich, Germany, 18-22 July 2011. pp. 108–117. IEEE (2011)
11. Francesco, N.D., Santone, A., Vaglini, G.: State space reduction by non-standard semantics for deadlock analysis. Sci. Comput. Program. 30(3), 309–338 (1998)
12. Jacob, G., Filiol, E., Debar, H.: Formalization of viruses and malware through process algebras. In: International Conference on Availability, Reliability and Security (ARES 2010). IEEE (2010)
13. Kinder, J., Katzenbeisser, S., Schallhart, C., Veith, H.: Detecting malicious code by model checking. Springer (2005)
14. Mercaldo, F., Nardone, V., Santone, A., Visaggio, C.A.: 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)
15. Milner, R.: Communication and concurrency. PHI Series in computer science, Prentice Hall (1989)
16. Song, F., Touili, T.: Pommade: Pushdown model-checking for malware detection. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM (2013)
17. Song, F., Touili, T.: Model-checking for android malware detection. Springer (2014)
18. Spreitzenbarth, M., Echtler, F., Schreck, T., Freling, F.C., Hoffmann, J.: Mobile-sandbox: Looking deeper into android applications. In: 28th International ACM Symposium on Applied Computing (SAC). ACM (2013)
19. Stirling, C.: An introduction to modal and temporal logics for ccs. In: Yonezawa, A., Ito, T. (eds.) Concurrency: Theory, Language, And Architecture. pp. 2–20. LNCS, Springer (1989)
20. Yang, T., Yang, Y., Qian, K., Lo, D.C.T., Qian, Y., Tao, L.: Automated detection and analysis for android ransomware. In: HPCC/CSS/ICESS. pp. 1338–1343. IEEE (2015)