RAMSES: Revealing Android Malware Through String Extraction and Selection

Lautaro Dolberg[1(✉)], Quentin Jérôme[1], Jérôme François[1,2], Radu State[1], and Thomas Engel[1]

[1] SnT - University of Luxembourg, 4, Rue Alphonse Weicker, 2721 Walferdange, Luxembourg
{lautaro.dolberg,quentin.jerome,jerome.francois, radu.state,thomas.engel}@uni.lu
[2] INRIA Nancy Grand Est, 615 Rue du Jardin Botanique, 54506 Vandœuvre-lès-Nancy, France
jerome.francois@inria.fr

Abstract. The relevance of malicious software targeting mobile devices has been increasing in recent years. Smartphones, tablet computers or embedded devices in general represent one of the most spread computing platform worldwide and an unsecure usage can cause unprecedented damage to private users, companies and public institutions. To help in identifying malicious software on mobile platforms, we propose RAMSES, an approach based on the static content stored as strings within an application. First we extract the contents of strings, transforming applications into documents, then using information retrieval techniques, we select the most relevant features based on frequency metrics, and finally we classify applications using machine learning algorithms relying on such features. We evaluate our methods using real datasets of Android applications and show promising results for detection.

Keywords: Android · Malware · Static analysis · Detection · Security

1 Introduction

The Android Operating System (OS) officially released in November 2007 is predicted to represent between 60 % and 70 % of smartphone operating systems in 2016[1]. It is also the main target of mobile threats for several reasons. Firstly, its growth provides a vast set of potential victims. Secondly, restriction and verification on published applications in the official market (Google Play Store) are limited and malicious applications have so still been published recently[2]. Finally, many third party markets are prone to store malware. Therefore, detecting Android malicious applications is of paramount importance. In this paper,

[1] http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats/a#phone-shipments, accessed on 04/30/2014.
[2] https://blog.lookout.com/blog/2013/04/19/the-bearer-of-badnews-malware-google-play/, accessed on 04/30/2014.

J. Tian et al. (Eds.): SecureComm 2014, Part I, LNICST 152, pp. 498–506, 2015.
DOI: 10.1007/978-3-319-23829-6_34

RAMSES relies solely on considering constant strings, which are extracted from the java code. Intuitively, malicious applications can differ from benign applications as they are performing different actions, like for instance accessing specific system files or connecting to uncommon remote services. It has been shown that communication channels used by popular applications (Facebook, Dropbox, etc.) can be misused using specific forged connection strings (URLs) [11]. Once strings are extracted, information retrieval methods are leveraged to compute metrics to a selected set of meaningful strings. Moreover, a widely abused feature consist in using the `Reflexion` API which allows the developer to call a method in specifying the name of the function as a `String` argument. We have also observed that accessing certain files is specific of a malware as highlighted in Sect. 3.

The rest of the paper is organized as follows: Sect. 2 review the basics of Android applications and related work. Section 3 explains the string based metrics which are used as input of decision algorithms in Sect. 4. Section 5 focuses on the evaluation. Section 6 discusses the method by highlighting the positive aspects and some limitations. Finally, Sect. 7 concludes the paper.

2 Background and Related Work

Android applications are coded in Java which are then compiled (class files) and converted by *dx* to Dalvik executable files (*dex* files). The Dalvik Virtual Machine (DVM for short) is then responsible for executing the bytecode on mobile devices similarly to the common Java Virtual Machine (JVM) on a computer. From a security perspective, each Android application is assigned to a set of permissions which is defined by the developer and has to be granted by the user when installing it. These can be abused in malicious applications since many users do not understand properly them, as discussed in [4].

According to several anti-virus vendors[3,4], the proportion of malicious applications for Android platform represents between 79 % to 99 % of reported malicous applications among all the mobile platforms. The authors in [13] characterize malware according how malicious payload is stored: either stored in the application itself or remotely deployed upon a unwanted user interaction. Complementary, malicious applications differ from their activations (bootstrapping events). A common practice for malicious applications is to escalate the privileges or to disrupt the proper functioning of the OS, to control the device remotely by an attacker. A dataset of 1260 malicious applications is introduced in [13]. Considering malware detection, Our approach differs from Andromaly [10] which uses a dynamic approach by extracting features during execution (CPU and Memory usage among others). Batyuk *et al.* used code disassembly to look at malicious API use in [1]. A similar approach based on `Permissions` and Control Flow Graphs (CFG) is presented in [8]. Walldroid [6] aims at monitoring

[3] http://thehackernews.com/2013/03/google-f-secure-can-say-that-anything.html accessed on 04/30/2014.

[4] http://www.securelist.com/en/analysis/204792255/Kaspersky_Security_Bulletin_2012_The_overall_statistics_for_2012 accessed on 04/30/2014.

and then blocking communications with malicious servers. A close approach to ours is [9] which also extracts strings but for application categorization (gaming, multimedia, etc.). On the contrary, RiskRanker [5] opted for a static approach aiming at extracting features from CFG to establish a risk score to a given application. Hence, to the best of our knowledge, we are the first to propose a static approach solely relying on string based metrics to identify Android malware.

3 Metrics

While a plethora of operands are available in Dalvik bytecode, only two of them caught our attention to characterize an Android application as strings:

- `const-string vAA, string@BBBB`
- `const-string/jumbo vAA, string@BBBBBBBB`

`string@` is the string index in a constant string pool and `vAA` is the destination register where the string will be loaded. The string index can be coded on two or four Bytes, this explains why there are two variants of this operand.

Because, we use strings to characterize applications, an application is represented as a document. We define formally a document as a list of $Term$. In this paper we will consider $Term$ and $String$ as equivalent. This extraction process is symbolized by the function $StringVariables$ which returns the list of terms of an application:

$$StringVariables \ : \ Application \rightarrow List(String) \tag{1}$$

Formally, this corresponds to a multiset, *i.e.* a set where the same element can appear multiple times. Assuming the application a_i, the corresponding document is a multiset of m terms:

$$d_{a_i} = \{t_1, \ldots, t_m\} \nRightarrow \forall t_i, t_j : t_i \neq t_j \tag{2}$$

From a set $A = \{a_0, \ldots, a_n\}$, containing n $Applications$, we define D_A as a multiset of all documents:

$$D_A := \{d_{a_i} : \forall a_i \in A\} \tag{3}$$

For sake of clarity, we illustrate such definitions with a simple example. Supposing a set A of two applications, each of them with three $Term$

$$\begin{aligned} A &= \{a_1, a_2\} \\ StringVariables(a_1) &= d_{a_1} = \\ &\quad [\text{“}system/etc\text{”}, \text{“}AndroidSDK\text{”}, \text{“}utf-8\text{”}] \\ StringVariables(App_2) &= d_{a_2} = \\ &\quad [\text{“}phone\text{”}, \text{“}AndroidSDK\text{”}, \text{“}system.data.void\text{”}] \end{aligned} \tag{4}$$

So the outcome of transforming set A into a multiset of documents D_A is:

$$\begin{aligned} D_A = \{ & \\ & [\text{“}system/etc\text{”}, \text{“}AndroidSDK\text{”}, \text{“}utf-8\text{”}], \\ & [\text{“}phone\text{”}, \text{“}AndroidSDK\text{”}, \text{“}system.data.void\text{”}] \\ \} & \end{aligned} \tag{5}$$

3.1 Information Retrieval

For sake of clarity, all applications are now considered as documents. For extracting relevant features, we derive common metrics for documents and terms [7]. Assuming a term t and a document $d \in D_a$, the term frequency (tf) of t is defined as:

$$\begin{aligned} &\text{tf} \ : \ Term \times Document \rightarrow \Re \\ &\text{tf(t, d)} = \frac{|\{t_i \in d : t_i = t\}|}{|d|} \end{aligned} \tag{6}$$

It is important to note that we are using multisets and so the numerator represents the number of times the term t occurs in the list of terms retrieved from a_i. For complexity reason, the idea is to select a subset of representative terms. Therefore, the rare terms are weighted stronger by using the inverse document frequency (idf) measure of the term t which is computed over a collection of documents D:

$$\begin{aligned} &\text{idf} \ : \ Term \times \{Document\} \rightarrow \Re \\ &\text{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d|} \end{aligned} \tag{7}$$

Finally, we obtain the weighted frequency also called term frequency-inverse document frequency ($tfidf$):

$$\begin{aligned} &\text{tfidf} \ : \ Term \times Document \times \{Document\} \rightarrow \Re \\ &\text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D) \end{aligned} \tag{8}$$

3.2 Most Relevant Terms

In theory, extracting all terms of all applications, and computing tf and $tfidf$ can be used to construct a complete set of features. However, this represents a large number of terms which then entails a large overhead for further analysis. In this paper, we extract the most relevant ones formalized as $Top_k^f(a)$, which, given an application a, returns the k's-terms with the highest scores for a given f (tf or $tfidf$). For example, the corresponding recursive definition using tf is:

$$
\begin{aligned}
&Top_k^{tf} \ : \ Document \rightarrow \{Term\} \\
&Top_1^{tf}(d) := \left\{ t : \max_{t \in d} \ tf(t,d) \right\} \\
&Top_k^{tf}(d) := \left\{ t : \max_{t \in d} \ tf(t,d) \right\} \cup Top_{k-1}^{tf}(d \backslash \{t\})
\end{aligned}
\tag{9}
$$

In the last line of Eq. (9), the expression $d\backslash\{t\}$ implies removing all the occurrences of the term t from the document d. As expected, Top_k^{tf} is a set of terms (not a multiset). By analogy, Top_k^{tfidf} is defined similarly but the collection of documents is also taken as an input for evaluating *idf*:

$$
\begin{aligned}
&Top_k^{tfidf} \ : \ Document \times \{Document\} \rightarrow \{Term\} \\
&Top_1^{tfidf}(d,D) = \left\{ t : \max_{t \in d} \ tfidf(t,d,D) \right\} \\
&Top_k^{tfidf}(d,D) = \left\{ t : \max_{t \in d} \ tfidf(t,d,D) \right\} \cup Top_{k-1}^{tfidf}(d \backslash \{t\})
\end{aligned}
\tag{10}
$$

Based on Eqs. (9) and (10), we retrieve the relevant by considering each term that is present in any Top_k^f for a given set of documents D:

$$
\begin{aligned}
&TopTerms_k^f \ : \ \{Document\} \rightarrow \{Term\} \\
&TopTerms_k^f(D) := \bigcup_{d_i in D} Top_k^f(d_i)
\end{aligned}
\tag{11}
$$

We obtain the two following sets $TopTerms_k^{tf}(D)$ and $TopTerms_k^{tfidf}(D)$.

4 Detection

Using the previous, we propose a scoring approach and a classifier based on machine learning to detect maclware. The assumptions are identical:

- a set of B known benign applications represented as documents $B = \{b_0, \ldots, b_B\}$
- a set of M known malicious applications represented as documents $M = \{m_0, \ldots, b_M\}$
- a set of U applications to be classified by our approach as benign or malware: $U = \{u_0, \ldots, u_M\}$

4.1 Scoring

The scoring approach runs in two steps. The learning procedure extracts the most relevant terms of benign and malicious applications using Eq. (11) and computes associated metrics (*tf* or *tfidf*). The testing stage looks for each of them in an application to classify, $u_i \in U$. For each term t belonging to $TopTerms_k^{tf}(B)$ and

appearing in u_i, the frequency of t is computed and added up over all documents in B which so results in a score. The same is applied to M. Formally, the *Score* function returns a numerical value from a given document and a collection of documents:

$$\begin{aligned} Score_k^{tf} &: \ Document \times \{Document\} \rightarrow \Re \\ Score_k^{tf}(q, D) &:= \sum_{t \in terms} \sum_{d \in D} \frac{tf(t,d)}{|D|} \\ \text{where} \qquad terms &= \{t : t \in q \ \wedge t \in TopTerms_k^{tf}(D)\} \end{aligned} \tag{12}$$

Similarly, this can be also calculated with *tfidf*, $Score_k^{tfidf}(q, D)$, assuming either B or M for calculating *tfidf* in the second line. Assuming a metric $f \in tf, tfidf$, u_i is marked as malicious if $Score_k^f(u_i, M) > Score_k^f(u_i, B)$, benign otherwise.

4.2 Machine Learning

Due the nature of the addressed problem, using machine learning seems appropriate. We consider a single metric (tf or $tfidf$) and the associated selected terms ($TopTerms_k^{tf}(D)$ and $TopTerms_k^{tfidf}(D)$) to construct the feature set. Therefore, for each application, we compute respectively either tf or $tfidf$ of these terms. While D represents a common dataset mixing malicious and benign applications, each instance is so labeled with a type of application during training and the testing stage has to predict it.

Some machine learning algorithms have been selected such that major types of approach are represented and also based on preliminary experiments: decision tree classifiers (*Random Forest*), rule-based approaches (*JRip*, *PART*) and function-based methods (*SGD*, *LibLINEAR* [12]).

5 Evaluation

In this section, only results based on *tfidf* are presented since `tf` results in an accuracy at least lower than 20 points compared to *tfidf*. Except when mentioned, k is set to 10. In addition, a 10 fold cross validation methodology has been employed.

Our evaluation employs the malware dataset from [13] This dataset is a recompilation of 1200 hand collected malware samples.

The market applications has been done by crawling and automatically retrieving applications about 25000 Google Android Market, supposed to be benign. We randomly sampled a subset to match the Malware dataset size.

5.1 Metric Analysis

As a preliminary experiment, our metrics are evaluated without being used for classification. We compute $TopTerms_k^{tf}(D)$ for both the malware dataset ($D = M$) and the benign application ($D = B$) with $k = 100$. In addition, we compute the intersection of *top* sets over all the couples of application of a given dataset defined as:

$$ComTerms(D) = \bigcup_{d_i,d_j \forall d_i \in D, d_j \in D, d_i != d_j} Top_k^{tf}(d_i) \bigcap Top_k^{tf}(d_j)\} \tag{13}$$

The constructed set represents all the words which are potential candidates to be representative of a specific type of applications (malicious or benign) because being at least shared between two of them. We derive the following facts:

$\lvert TopTerms_{100}^{tf}(M)\rvert$	$\lvert TopTerms_{100}^{tf}(B)\rvert$	$\lvert ComTerms_{100}^{tf}(M)\rvert$	$\lvert ComTerms_{100}^{tf}(B)\rvert$
3538	9671	102	945

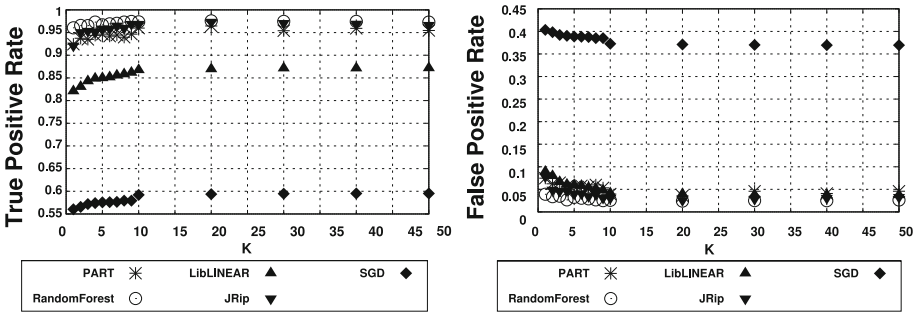
Such results highlight that selected terms by our method are helpful to characterize the type of applications. In particular, the *top* sets of malicious applications is highly smaller than for normal applications, even if the size of each dataset is equal. Considering the number of shared terms between at least two applications, the number is drastically decreased for both datasets but in different orders of magnitude. For benign applications, the number is divided by about ten while it is divided by more than 30 % for malware. On the first hand, this means that malicious applications could be characterized by a smaller number of strings. On the other hand, it is representative of the frequent use of common strings in normal applications.

5.2 Scoring

The scoring approach computes two scores for each tested application: one based on malicious applications, $Score_k^f(u_i, M)$, one based on benign applications, $Score_k^f(u_i, B)$, both with $k = 10$. As shown in Table 1, there exist applications (malware). However, the separation is not always evident as highlighted by an accuracy around 65 %.

Table 1. Scoring classifier performance (TF-IDF) in percentage

Type	Bening	Malware
$Score_k^f(u_i, B) > Score_k^f(u_i, M)$	65	24
$Score_k^f(u_i, B) < Score_k^f(u_i, M)$	30	69
$Score_k^f(u_i, B) = Score_k^f(u_i, M)$	5	7



(a) True Positive Rate (b) False Positive Rate

Fig. 1. Malware identification results (TF-IDF)

5.3 Machine Learning

As more advanced techniques, machine learning methods are expected to be more accurate. We have first assessed the value of k for good classification performances since k controls the number of terms (classification features) in $TopTerms_k^{tf}(D)$ and $TopTerms_k^{tfidf}(D)$. So, it is highly important to reduce the number of features and so k in order to limit the overhead of the classification algorithm. True and false positive rates (TPR and FPR) are shown in Fig. 1 when k varies. This highlights the viability of classifying classification using only embedded strings. In particular, Random Forest [2] is the best classifier with the highest TPR with the lowest FPR. Naturally, when more terms are used (increasing k), the performances are better but having k higher than 10 does not improve results significantly whereas $k = 10$ provides good results with $TPR = 0.97$ and $FPR = 0.025$.

6 Discussion

Our method solely relies on strings easily extracted from applications. However, in case of armed malware or applications having encrypted or packed code and data, this would limit its practicability. Nevertheless, most of application markets dont accept these applications. It is also possible to imagine a malware developer including unused strings or dead code sections, in particular to add strings which are usually present in benign applications. A solution will be to consider only a malicious dataset during the training and apply one-class classification. Furthermore, code deobfuscation is an unresolved problem by the community, even assuming encrypted strings [3]. Such a technique could be used as a preprocessing step of our method. Another option for the attacker is to divide the malware into multiple programs. Even if it is not well widespread yet, our method can easily cope with such an issue by merging the set of strings of multiple applications.

7 Conclusion

RAMSES is able to characterize Android malware based on constant strings of the Dalvik bytecode and information retrieval techniques. Its main advantage is that it can be used as a statical analysis tool without having to run suspicious or untrusted applications. However, the proposed work could also be envisioned as a first analysis to pre-select applications which need an in-depth analysis. As a future work, we will move towards a collaborative approach based on user feedback.

Acknowledgement. The Authors would like to thank the National Research Fund of Luxembourg (FNR) for providing financial support trought CORE 2010 MOVE Project.

References

1. Batyuk, L., Herpich, M., Camtepe, S.A., Raddatz, K., Schmidt, A.D., Albayrak, S.: Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within android applications. In: International Conference on Malicious and Unwanted Software, pp. 66–72 (2011)
2. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
3. Bremer, J.: Automated analysis and deobfuscation of android apps & malware. In: AthCON (2013)
4. Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D.: Android permissions: user attention, comprehension, and behavior. Technical report, EECS Department, University of California, Berkeley, February 2012
5. Grace, M., Zhou, Y., Zhang, Q., Zou, S., Jiang, X.: Riskranker: scalable and accurate zero-day android malware detection. In: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, MobiSys (2012)
6. Kilinc, C., Booth, T., Andersson, K.: Walldroid: Cloud assisted virtualized application specific firewalls for the android os. In: Trust, Security and Privacy in Computing and Communications (TrustCom). IEEE (2012)
7. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, New York (2008)
8. Sahs, J., Khan, L.: A machine learning approach to android malware detection. In: Intelligence and Security Informatics Conference (EISIC). IEEE (2012)
9. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P.: On the automatic categorisation of android applications. In: 2012 IEEE Consumer Communications and Networking Conference (CCNC). IEEE (2012)
10. Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y.: Andromaly: a behavioral malware detection framework for android devices. J. Intell. Inf. Syst. **38**(1), 161–190 (2011)
11. Wang, R., Xing, L., Wang, X., Chen, S.: Conference on computer and communications security (ccs). In: Unauthorized Origin Crossing on Mobile Platforms: Threats and Mitigation. ACM (2013)
12. Witten, I.H., Frank, E., Hall, M.A.: Data Mining: Practical Machine Learning Tools and Techniques, 3 edn. Morgan Kaufmann, San Francisco
13. Zhou, Y., Jiang, X.: Dissecting android malware: characterization and evolution. In: Symposium on Security and Privacy. IEEE (2012)