2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems

Pinpointing Mobile Malware Using Code Analysis

Omer Tripp
IBM Watson Research Center
otripp@us.ibm.com

Marco Pistoia
IBM Watson Research Center
pistoia@us.ibm.com

Pietro Ferrara
JULIA S.R.L.
pietro.ferrara@gmail.com

Julia Rubin
MIT
mjulia@csail.mit.edu

ABSTRACT

Mobile malware has recently become an acute problem. Existing solutions either base static reasoning on syntactic properties, such as exception handlers or configuration fields, or compute data-flow reachability over the program, which leads to scalability challenges.

We explore a new and complementary category of features, which strikes a middleground between the above two categories. This new category focuses on security-relevant operations (communcation, lifecycle, etc) — and in particular, their multiplicity and happens-before order — as a means to distinguish between malicious and benign applications. Computing these features requires semantic, yet lightweight, modeling of the program's behavior.

We have created a malware detection system for Android, MassDroid, that collects traces of security-relevant operations from the call graph via a scalable form of data-flow analysis. These are reduced to happens-before and multiplicity features, then fed into a supervised learning engine to obtain a malicious/benign classification. MassDroid also embodies a novel reporting interface, containing pointers into the code that serve as evidence supporting the determination.

We have applied MassDroid to 35,000 Android apps from the wild. The results are highly encouraging with an F-score of 95% in standard testing, and >90% when applied to previously unseen malware signatures. MassDroid is also efficient, requiring about two minutes per app. MassDroid is publicly available as a cloud service for malware detection.

CCS Concepts

•Security and privacy → Usability in security and privacy; •Theory of computation → *Program analysis;*

Keywords

malware detection, static analysis, machine learning, trace features, classification

MobileSoft'16 May 16-17 2016, Austin, TX, USA

ACM ISBN 978-1-4503-4178-3/16/05...

DOI: http://dx.doi.org/10.1145/2897073.2897707

1. BACKGROUND & MOTIVATION

Malware is among the main threat types in the mobile space. Malicious software has the purpose of stealing data, damaging the device, blackmailing or annoyting the user, etc. The attacker either defrauds the user into installing the app or accesses the device remotely, without the user's authorization, by exploiting a vulnerability (e.g. if the device is rooted). The malware family includes Trojans, worms, botnets and viruses.

Static malware detection is an active area of research, where the goal is to design techniques that are scalable and robust. By scalability we mean that reasoning based on the given features is performant and applicable to industry-scale applications. Robustness means that the features are hard to bypass, and would require the malware designer to invest significant time and effort to evade detection.

To meet the requirement for scalability, several past solutions have utilized lexical features to guide their reasoning [7, 8, 1]. These include e.g. the syntactic structure of exception handlers or the app's configuration (e.g., the permissions specified by an Android app in its manifest file).

For robustness, another set of features — based on data-flow reachability between security-relevant platform APIs [3] — has recently been proposed. These capture behaviors of the program that cannot be disguised via simple syntactic manipulations (since data still flows between the platform APIs even if client code is obduscated). However, making a solution based on such features scalable entails nontrivial engineering challenges.

While both of these classes of features are useful, they each emphasize one advantage at the expense of the other. Syntactic features are geared toward simplicity and performance, whereas data-flow reachability offers robustness at the cost of limited scalability.

2. OUR APPROACH

We explore a third and complementary category of features, which — in our empirical experience — offers high discriminative power w.r.t. malware detection, and at the same time strikes a more balanced performance/robustness tradeoff compared to the existing feature categories. In this category, we focus on security-relevant operations. These include logging, communication (such as inter-application, SMS and HTTP messages), access to sensitive identifiers (such as the user's social profile), lifecycle events (such as switching views or restarting the program), data conversions (e.g., from location to address), etc.

The main insight underlying our approach is that certain

sequences of security-relevant operations are indicative of malware, and can thus be used to distinguish between malicious and benign applications. For example, malicious applications often exploit the user by sending multiple SMS messages to premium numbers or services. Benign applications, in contrast, rarely send SMS messages one after the other in general.

Driven by this insight, we define two families of features that can be extracted by analyzing mobile application binaries. **Happens-before features** are boolean predicates over tuples $(o_1, \ldots, o_n)$ of operations, such that if there exists an execution trace involving all the operations o_i, and in the order specified in the tuple, then the feature holds true. **Multiplicity features** are ternary unary predicates with possible values 0, 1 and > 1. Given operation o and execution trace τ, the respective multiplicity feature of o, denoted by $m[o]$, has the value 0 if o is absent from τ; 1 if o appears in τ exactly once; or > 1 otherwise.

We have implemented the MassDroid system for detection of Android malware atop trace features. MassDroid is parameterized by a specification S of security-relevant operations. In the case of Android, these include Android lifecycle events [2] as well as well-known privacy and integrity sources and sinks [9].

Given input program $\mathbb{P}$ and specification S of security-relevant operations, MassDroid creates a call-graph representation G of $\mathbb{P}$. It then exhaustively enumerates all feasible (static) execution traces τ of $\mathbb{P}$, collecting a regular approximation of the involved security-relevant operations (per S). These are then reduced to happens-before and multiplicity features.

The features are combined into a feature vector (by aggregating over all traces, as explained above), which MassDroid feeds into a supervised machine-learning classifier. The classifier — utilizing the popular support-vector machine (SVM) algorithm [4] — is trained offline over both malicious and benign applications.

The static analysis that MassDroid performs is a lightweight form of data-flow analysis — to compute regular traces — that is far cheaper than data-flow reachability. As such, there is no need to statically track or approximate argument values, memory manipulations or other aspects of the program's state that are expensive to model [6]. To perform the analysis efficiently, we exploit the fact that it distributes over traces, thereby falling within the scope of the IFDS framework for interprocedural distributive data-flow analysis [10] by means of abstract interpretation [5].

To enable validation of, and action based upon, its benign/malicious determination, MassDroid further reports evidence in support of the determination. This is done in two steps. First, as part of offline training, MassDroid records features that have high discriminative power (either occurring in most malicious apps and almost none of the benign apps or vice versa). Then, while statically analyzing an input app, MassDroid stores the code positions corresponding to the operation traces it computes. Finally, when the machine-learning engine makes a determination, traces that model discriminative features consistent with the determination are highlighted in the report.

3. EXPERIMENTAL EVALUATION

For the question of how well MassDroid can distinguish between malicious and benign samples, we composed a dataset of 19,000 applications, 15,000 of which being benign Playdrone applications and the remaining 4,000 being malicious samples drawn randomly. This 15 : 4 ratio leans heavily toward benign applications, which are indeed more prevalent than malicious apps, but at the same time malware is well represented to avoid classifier proneness to false negatives.

We divided the resulting dataset into training and testing sets, such that (roughly) $\frac{2}{3}$ of the apps — selected at random — were used for training, and the remaining 6,431 apps for testing. As such, throughout testing we also maintain a ratio of 15 : 4 between benign and malicious apps, which reflects the greater proportion of benign apps in the wild while at the same time evaluating the classifier on a significant number of malware samples. The classification results we obtained are as follows:

TPs	FPs	FNs	precision	recall	F-score
5,922	178	331	94.7%	97.1%	95.9%

These results validate the efficacy of MassDroid in detecting malicious applications. In particular, MassDroid's recall is above 97%, and above its precision, which is consistent with our design goal of treating missed malware as more severe than misclassification of a benign app as malicious.

Breakdown of the overall time spent by MassDroid on classification of the test applications yields an average of 62.91 seconds and 105.83 seconds to compute histories for malicious and benign applications, respectively. We conjecture that the significant gap is due to the fact that benign apps often feature richer and more diversified behaviors compared to malware, and are therefore typically larger and more complex.

4. REFERENCES

[1] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In *NDSS*, 2014.

[2] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, and P. McDaniel. FlowDroid: Precise Context, Flow, Field, object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. In *PLDI*, 2014.

[3] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden. Mining apps for abnormal usage of sensitive data. In *ICSE*, 2015.

[4] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, 1998.

[5] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, 1977.

[6] M. Hind. Pointer analysis: haven't we solved this problem yet? In *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis For Software Tools and Engineering, PASTE'01, Snowbird, Utah, USA, June 18-19, 2001*, pages 54–61, 2001.

[7] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie. Whyper: Towards Automating Risk Assessment of Mobile Applications. In *USENIX Security*, 2013.

[8] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen. AutoCog: Measuring the Description-to-permission Fidelity in Android Applications. In *CCS*, 2014.

[9] S. Rasthofer, S. Arzt, and E. Bodden. A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks. In *NDSS*, 2014.

[10] T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *POPL*, 1995.