A Collaborative Framework for Generating Probabilistic Contracts

Fabio Martinelli, Andrea Saracino, Daniele Sgandurra
Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche, Pisa, Italy
name.surname@iit.cnr.it

Alessandro Aldini
Dipartimento di Scienze di Base e Fondamenti
Università di Urbino Carlo Bo, Italy
name.surname@uniurb.it

***Abstract*—We propose a collaborative framework for generating probabilistic contracts for Android smartphones aimed at detecting repackaged applications. To this end, a network of users sends to the application server the sequences of actions that represent the usage profile of the application. Then, the application server generates a contract from this set of traces. Contracts are represented through clustered probabilistic automata. At run-time, a monitoring system on the smartphone verifies the compliance of the running application against the contract through the Pearson's Chi Squared test. In the preliminary tests, the proposed framework has been able to detect repackaged applications whose behavior is strongly similar to the original application but hide malware.**

***Keywords*—Collaborative Framework; Probabilistic Contract; Android; Malware; Repackaging;**

I. Introduction

Mobile markets distribute a plethora of applications for mobile devices. Official and unofficial markets often provide several versions of the same application. Unofficial markets, in particular, may offer free versions of applications that on the official markets have to be paid. Some malicious developers exploit these channels to distribute malware, usually for open-platform Smartphones such as Android. Usually, malware applications are modified versions of genuine applications that contain malicious code to performs bad operations. This happens while the user normally exploits the genuine part of the application only and, hence, the bad behavior passes unnoticed. These applications are called *trojanized* or *repackaged* applications.

Repackaged applications can be identified verifying their compliance with the *contract* built for the genuine version of the application. A contract is a document that describes the operations that an application can perform during its execution. A repackaged application behaves in a way non-compliant with the contract of the original application, hence a misbehavior can be identified if a proper contract has been generated for the original application.

In this paper we present a collaborative framework that exploits Android system call analysis and semantic clustering to build probabilistic contracts. These dynamically-built contracts are behaviorally-based and different behaviors of various users are merged together to generate a contract that is representative of the application. At run-time, i.e. after the contract has been generated, the framework analyzes the behavior of untrusted applications by matching their behavior against the associated contract, to differentiate genuine applications from repackaged ones. The system has been developed and tested on Android smartphones.

This work has been funded by the European Union FP7 under grant no 256980 (NESSoS) and no 257930 (Aniketos) and PRIN Security Horizons.

The main contributions of this paper are:

- the description of a collaborative framework that collects data from a group of collaborative users to build probabilistic contracts;
- a verification mechanism based upon the probability distribution of the longest execution paths;
- a method to test the probabilistic compliance of a behavior against a contract based on Chi Squared test.

The remainder of the paper is organized as follows. Section II discusses the collaborative framework to collect traces of usage profile and the method to describe the applications' behavior through probabilistic automata. Section III describes the contract matching technique to detect misbehaviors and reports some preliminary experiments to prove the effectiveness of the proposed approach. Section IV reports some related works. Finally, Sect. V concludes and proposes some extensions.

II. Contract Generation

Before generating a contract, the framework requires several traces of an application representing the usage profile. Then, from the traces sent by a network of collaborative users, the framework generates the probabilistic contract.

A. Contract

A *contract* is a document that describes the expected behavior of an application. An application α is *compliant* with the contract γ ($\alpha \models \gamma$) when all the actions effectively performed by the application are included in the contract. A contract can be defined using information that can be computed either statically or dynamically. In the static approach, the contract can be built by learning some properties from, e.g., the source code. However, it may be impossible to know in advance some properties of the code, e.g. behavior of the application that depends about some inputs. In the dynamic approach, a contract can be defined by exploting the information learnt from the application's executions, e.g. by monitoring some

978-1-4673-6404-1/13/$31.00 ©2013 IEEE

executions to extract their behavior. Here, we will focus on the dynamic approach, since it is more suitable to represent those applications whose behavior depend upon user inputs.

Contracts defined using dynamically-generated behaviors may be probabilistic and, since they are built upon execution traces, it is possible to compute the probability that each action is performed, including a *quantitative* information in the contract. We say that an application α is compliant with a probability threshold ξ to the probabilistic contract γ ($\alpha \models_\xi \gamma$) if any action performed by α is in the contract and happens with a probability greater or equal to $\theta = 1 - \xi$.

B. Collaborative Traces Acquisition

In the proposed framework, an application *trace* is a sequence of system calls. During its lifetime, a process issues several system calls that composes a trace. If we transform this trace into an oriented graph, where each system call is a node, and edges represents the transition from a system call into the next one, then it is possible to provide a signature of the application behavior. To this end, we have developed an application for Android devices that hooks the system calls performed by an application and builds a sequence that is sent to a central server. Then, the server applies the algorithm (described in the next Subsection) and builds the *execution multi-graph* representing a *trace*, and the automaton which describes a *partial contract*.

If we only consider the traces recorded by a single user, then the contract would be partial since a user rarely explores all of the possible application states and this would not be sufficient to generate a consistent contract. To generate a proper contract, more traces generated by several users need to be merged. For this reason, the framework exploits a collaborative approach where each user that installs the application sends the execution multi-graph of the app to the central application server, which builds the contract for the application by merging all the execution graphs.

A trace is recorded from the time when the application is started and ends when the application is closed, or after a sufficiently long time of the app's execution. The resulting multi-graph is sent to the central server, which holds a database of applications and their contracts. To be more specific, when a user sends a multi-graph, the server verifies if the application is already in the database. If the application is missing, a new record is added, and the probabilistic automaton, i.e. the contract, is generated from the multi-graph and it is returned to the user. If the application is already in the database, the multi-graph is merged with the one already stored. The operation of merging two multi-graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, returns a new graph $G_3 = (V_3, E_3)$ with $V_3 = V_1 \cup V_2$ and $E_3 = E_1 \cup E_2$. From the resulting graph G_3 the probabilistic automaton is recomputed and sent back to all the users that participated in the contract generation or that simply own the application. Notice that these operations can be clustered and the updated contract may be re-sent to users on a daily basis. The application installed on the device, which checks the compliance of the contract, verifies the authenticity of the contract by checking the signature of the server, whose public key is shipped with the application. We assume that all the traces used to generate the contract are trustworthy, i.e. the users are not malicious and the applications they are analyzing are not repackaged.

C. Traces Analysis and Contract Generation

To properly represent the contract using the traces produced collaboratively, we exploit the notion of *ActionNode* (see [1] for its detailed explanation), which is a cluster of related system calls, i.e., the graph of system calls that are consecutively issued in the trace and that are bound by some relation and that form an *action* (a high-level operation). As an example of ActionNode, consider the sequence of system calls: `open(A) - read(A) - close(A)`, where A is the filename. This ActionNode represents at high level the action of reading data from file `A`: this action requires that, firstly, the file has to be opened, then data is read and, finally, the file is closed. In the following, we consider only the system calls that act on files, namely the `open`, `read`, `write`, `close`, `ioctl`.

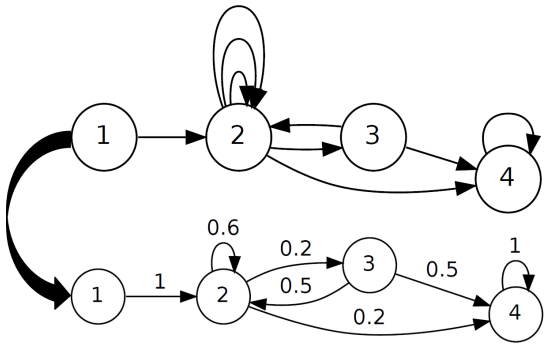

Figure 1: From Multi-Graph to Probabilistic Automaton

Using this notion, the system call traces are represented by a *multi-graph* of ActionNodes. Formally the multi-graph is a directed graph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is the set of ActionNodes that describe the high-level operations performed by a specific application, and E is the set of edges $e_{i,j}$ outgoing from the node i and entering the node j. Being a multi-graph, there can be both multiple occurrences of each edge $e_{i,j}$ and edges insisting on a same node, that is $i = j$. Then, from the multi-graph an automaton is built (see, e.g., Fig.1). The automaton $A = (V, T)$ contains the same nodes V of the multi-graph, but multiplicities of edges are not taken into account, that is, the edges $e_{i,j}, e_{i',j'}$ where $i = i', j = j'$ are collapsed into a single edge $e_{i,j}$. On the other hand, multiplicities of edges are used to compute the related probabilities in A. Formally, $T \subseteq E \times]0; 1]$ is such that:

$$(e_{i,j}, p_{i,j}) \in T \text{ iff } \exists e_{i,j} \in E \wedge p_{i,j} = \frac{mul(i,j)}{mul(i)}$$

where $mul(i, j)$ is the multiplicity of $e_{i,j}$ in E and $mul(i)$ is the number of outgoing edges from v_i in the multi-graph. This transformation induces a probability distribution for each node, i.e., $\forall v_i : \sum_{v_j \in V} \{p_{i,j} \mid (e_{i,j}, p_{i,j}) \in T\} = 1$. Hence,

each edge in T departing from a given node v_i is enriched with the probability that it is traversed by taking into account the number of occurrences of each edge in E departing from the same node v_i.

At run-time, the compliance of the application behavior with the contract will be evaluated by measuring the frequency of certain finite observations with respect to the probability distributions that are associated with such observations in the contract. The finite observations to which we associate probability distributions are given by the finite *longest acyclic paths* in the automaton. Given a node v_i, the longest acyclic paths starting from v_i are all the paths that can be followed from v_i towards all the other nodes in the graph, without traversing more than once each node in the path, except possibly for the last one. Formally, a *path* is a sequence $v_{i_1} \dots v_{i_n}$ such that $e_{i_j,i_{j+1}} \in E$ with $1 \leq j < n$ and a *longest acyclic path* is a path $v_{i_1} \dots v_{i_n}$ such that $\forall v_{i_j}$, $1 \leq j < n$, there does not exist $k \neq j$, $1 \leq k < n$, such that $v_{i_j} = v_{i_k}$. As usual, the probability of a path $v_{i_1} \dots v_{i_n}$ is computed as the product of the probabilities $p_{i_j,i_{j+1}}$ of the edges $e_{i_j,i_{j+1}}$ (with $1 \leq j < n$) forming the path. Since we deal with probability distributions and finite paths, for each node v_i extracted from the contract, we derive a probability distribution over the set of longest acyclic paths starting from v_i such that the probabilities associated with all these paths sum up to 1. In particular, we compute these probability distributions for each node of the automaton that is not a traversing node, i.e., a node that has not ears (except the starting node). In the end, the server builds and signs (with its private key), the contract, which is a pair composed of the automaton and the probability distributions.

III. Contract Compliance and Tests

Once the contract has been built, it can be used to verify the compliance of different versions of the same application on distinct Smartphones where these versions have been installed. A non-compliant application is an application that shows a different behavior from the one declared in the contract. More specifically, an application is non compliant when performs one or more operations that are not included in the contract or when an operation (or a sequence of operations) is not compliant with the probability distribution described in the contract.

Let C be the probabilistic contract of an application A. We want to verify the compliance of A', a different version of A, against C. To this end, the framework monitors the behavior of A' by progressively building the ActionNodes and extracting the occurrences of the longest acyclic paths. Then, the framework uses the *Pearson's Chi Squared Test* [2] to test the consistency of the behavior of A' with respect to the probability distributions associated with C. The Pearson's Chi Squared Test is used in statistics to verify if a sample is statistically consistent with respect to a known probability distribution. In practice, given a realization belonging to a set of events whose probabilities is known, the Chi Squared test states if the realization is consistent or not with the probability distribution. Hence, the events generated by A' represent the statistical sample, whilst the contract describes the known distribution.

For each node v_j of the contract of A that is not a traversing node, all the longest acyclic paths are listed. Let us suppose their number is n_j. At run-time, the framework incrementally builds the paths for A' by recording the longest acyclic paths and using the same algorithm described for generating the contract. For each node, the framework keeps track of the occurrences of the longest paths observed from this node in order to compute the related probabilities O_i. Then, it derives from the contract the expected probabilities E_i. Notice that, a node with no probability distribution in the contract clearly means that the behavior is non-compliant with the contract. Then, the chi-squared score χ^2 is computed, for each starting node v_j, according to the following formula $\chi^2 = \sum_{i=1}^{n_j} \frac{(O_i - E_i)^2}{E_i}$. To verify the chi-squared null hypothesis, i.e., the behavior of A' has a distribution consistent with the one described in the contract C, the test statistic is drawn from a chi-squared distribution with one degree of freedom. If the computed probability is higher than conventional criteria for statistical significance the null hypothesis is not rejected, i.e., the behavior is compliant with the contract and the application should not be considered repackaged.

A. Preliminary Tests

We report an example taken from some preliminary tests that we have performed. Figure 2 shows the contract built merging several traces of the application `TicTacToe`, a sample application provided with the Android SDK. Table I shows some longest acyclic paths, starting from node 5 and their probabilities. All the traces have been collected from several different users, which tried to explore all the functionalities of the application, to compute a representative contract. Then, we have built a repackaged version of this application by updating the application code by injecting into it a piece of malware code to send an SMS message each time the user opens the menu to change the game skin. Then, we have run the modified program several times.

As an example, by computing the χ^2 test on the distribution of longest acyclic path outgoing from the node 5, which is the action node `read(A) - read(A)`, the observations resulted non-consistent with the probability distribution. The main issue has been given by the path 5-3-3, where 3 is the action node `ioctl(A) - ioctl(A)`, whose probability in the contract is $4 \cdot 10^{-4}$. Given a set of about 1000 observations of path outgoing from node 5, with the repackaged version of the app, we have found more than 120 occurrences of the path 5-3-3, obtaining a model that strongly differs from the expected probability distribution.

IV. Related Work

System call analysis has been used in several system to monitor and detect misbehaving. [3] proposes *Crowdroid*, an IDS that is based on the number of system calls issued by an

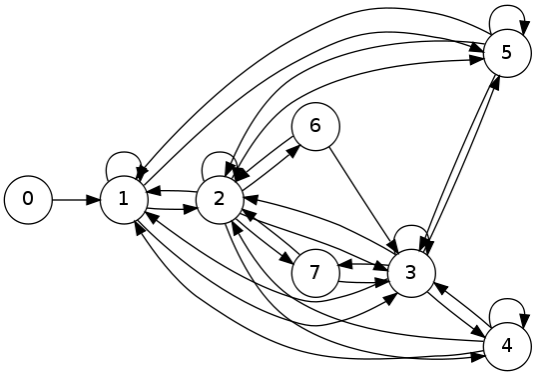

Figure 2: Probabilistic Contract of TicTacToe

Table I: Some Longest Paths From Node 5

Path	Probability
5-3-1-1	0.00005476
5-3-1-3	0.0002
5-3-1-5	0.000001
5-3-2-2	0.00188
5-3-2-3	0.0034
5-3-2-5	0.00002
5-3-3	0.0004
5-3-4-1	0.0000012
5-3-4-3	0.00014
5-3-4-4	0.000189
5-3-5	0.0031
5-3-7-3	0.0000156

application. Misbehaviors are identified by applying computational intelligence techniques. Another system that exploits system calls and computational intelligence is presented in [4], which is an anomaly-based intrusion detection system that, differently from Crowdroid, monitors the system globally, but it may not be able to detect some trojanized application if their behavior faithfully represents the good ones. Some Android security frameworks try to protect the system by monitoring the communication level and defining security policies. One of these systems is presented in [5], which allows the definition of context based security policies. Analysis of system calls with Markov Models have formerly been performed on other operating systems. In [6] a scheme for intrusion detection is proposed. This system exploits system calls and hidden Markov models and is able to detect efficiently denial of service attacks. [7] presents another system based upon system calls and Markov models to detect intrusions. This system analyzes the arguments of the system calls but is oblivious of the system call sequence. System call sequence and deterministic automata have been used in [8] to detect anomalies, which are detected when system call sequences differ from an execution trace known to be good. A model to capture reliability and availability properties of stochastic systems using probabilistic contracts has been presented in [9], [10]. Finally, the chi-squared test has been used to detect anomalies in several field. An application on network traffic analysis is presented in [11].

V. Conclusions

In this paper we have presented a collaborative framework to build mobile application contracts. The framework exploits system call analysis to create an automaton and statistical inference to detect repackaged applications. We have built a first prototype of both the application running on the device and the collaborative framework and tested the effectiveness of the method through some preliminary experiments.

We are planning as future work the inclusion of a distributed reputation model to loosen the assumption that the collaborative users are trustworthy when creating the contract. The idea is to give a different probabilistic weight to the received traces, based upon the user reputation and the similarity of the received traces with those already stored on the server. Finally, the prototype will be updated to improve its usability and performances.

References

[1] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra, "A framework for probabilistic contract compliance," Istituto di Informatica e Telematica, CNR, Tech. Rep. IIT TR-03/2013, Feb 2013. [Online]. Available: http://www.iit.cnr.it/sites/default/files/trpicard.pdf

[2] R. L. Plackett, "Karl Pearson and the Chi-Squared Test," *International Statistical Review / Revue Internationale de Statistique*, vol. 51, no. 1, pp. 59–72, 1983.

[3] U. Z. I. Burguera and S. Nadijm-Tehrani, "Crowdroid: Behavior-Based Malware Detection System for Android," in *SPSM '11*. ACM, October 2011.

[4] G. Dini, F. Martinelli, A. Saracino, D. Sgandurra, "MADAM: A Mult-Level Anomaly Detector for Android Malware," in *6th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security, MMM-ACNS 2012, St. Petersburg, Russia*, vol. 7531 - 2012. Springer Verlag, October 2012.

[5] M. Conti, V.T.N. Nguyen,B.Crispo, "CRePE: context-related policy enforcement for android," in *ISC'10 Proceedings of the 13th international conference on Information security* . Springer-Verlag, 2010, pp. 331–345.

[6] X.A. Hoang, J. Hu, "An efficient hidden Markov model training scheme for anomaly intrusion detection of server applications based on system calls ," in *12th IEEE International Conferecence On Networks, ICON 2004*, vol. 2. IEEE, November 2004, pp. 470 – 474.

[7] F. Maggi, M. Matteucci, S. Zanero, "Detecting Intrusions through System Call Sequence and Argument Analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, October-December 2010.

[8] A.P. Koresow, "Intrusion detection via system call traces," *Software*, vol. 14, no. 5, 1997.

[9] B. Delahaye, B. Caillaud, and A. Legay, "Probabilistic contracts: a compositional reasoning methodology for the design of systems with stochastic and/or non-deterministic aspects," *Formal Methods in System Design*, vol. 38, no. 1, pp. 1–32, 2011.

[10] A. L. B. Delahaye, B. Caillaud, "Probabilistic contracts : A compositional reasoning methodology for the design of stochastic systems." in *Proc. 10th International Conference on Application of Concurrency to System Design (ACSD), Braga, Portugal*. IEEE, 2010.

[11] N. Ye and Q. Chen, "An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems," *Quality and Reliability Engineering International*, vol. 17, no. 2, pp. 105–112, 2001. [Online]. Available: http://dx.doi.org/10.1002/qre.392