On Probabilistic Application Compliance

Antonio La Marra, Fabio Martinelli, Andrea Saracino
Istituto di Informatica e Telematica, Consiglio Nazionale delle Ricerche, Pisa, Italy

Alessandro Aldini
Dipartimento di Scienze Pure e Applicate, Università di Urbino, Urbino, Italy

Abstract—The Security-by-Contract is a paradigm developed to offer a secure environment in which mobile applications can be executed by respecting the security policies of interest. Especially in the Android Apps marketplace, establishing precisely the expected secure app behavior is typically a complex operation that is prone to approximations. Hence, it is worth considering extensions of purely functional approaches that allow the security relevant actions to be quantitatively assessed. This also opens the possibility to balance the application of (expensive) enforcement mechanisms with the security guarantees. With these objectives in view, in this paper we define a probabilistic extension of the Security-by-Contract model, and we show its impact in real-world scenarios through the analysis of several practical Android applications.

Index Terms—security by contract; Android applications; probabilistic models; enforcement;

I. INTRODUCTION

Mobile devices such as smartphones, tablets, and wearables, are increasing the expressiveness and popularity of mobile computing. This is due not only to the augmented computing power and ubiquitousness, but also to the enormous growth of the available applications. Mobile apps are versatile, rely strongly on the high connectivity offered by these devices, and are designed to cover categories of use ranging from life fun to business. For instance, their use in corporate environments is growing to enhance engagement, promotion, and customer support.

Security represents a critical issue for the market of mobile apps, which are exposed to many types of cyber-attacks. Malicious developers strive to design mobile apps damaging both users and devices, by using, e.g., Trojan horses, which may cause corporate (or personal) data (or money) theft and leakage. A recent study sponsored by IBM [1] reveals that companies test less than 50% of the mobile apps they build, while a company out of three never conducts security tests before the app deployment. On the other hand, paradigms like the Bring-Your-Own-Device are adopted by an increasing number of companies that allow employees to use business apps on their personal devices. Similarly, nowadays a large amount of users feel confident in downloading and using apps that manage their credentials and personal information, like financial data or patient records, without paying attention to the risks deriving from the use of potential repackaged apps.

In this setting, it is standard to use security models based on the specification of contracts establishing what kind of actions the app can execute. The underlying semantics is either based upon trust relationships or upon statements of purpose. In the former case, the users run a mobile app because they trust the app provider, essentially leading to a security model based on the “all or nothing” policy, that is either the app is trusted and, therefore, allowed to do anything, otherwise it is not installed. In the latter case, the app provider declares the security relevant actions that could be performed by the app, so that the users can decide whether to run the app or not, by possibly restricting the app access rights. However, in such a case typically the semantics is too-coarse grained and scarcely user-friendly. For instance, in the Android system, security relevant actions are declared through permissions, which are difficult to understand for average users.

For these reasons, fully automated, contract-based intrusion detection systems are proposed to facilitate and make it flexible the task of controlling the behavior of the installed mobile apps. In particular, the Security-By-Contract approach is based on a mobile, formal notion of contract that accompanies the app and has a twofold objective. On one hand, it is used to establish whether the app behavior is compliant with the declared scope as targeted by the app builder. On the other hand, it represents the base for checking whether the app is secure with respect to specified policies. The validation of such a twofold compliance check is strictly related to the completeness and exactness of all the parameters at hand: the actual observations stating the app behavior, the statements describing the respected contract, and the specification of the security policies. However, in real-world scenarios, it can be difficult to estimate precisely some of these parameters while, at the same time, fluctuations of non-functional features may have an effect upon the compliance results. Therefore, in this paper we propose a quantitative approach to the application of the Security-By-Contract paradigm that is based on probabilistic information. In this setting, statistical inference analysis and approximation techniques are used to estimate the levels of compliance mentioned above. Only in the case in which such run-time quantitative checks are not satisfied by the monitored app, adequate enforcement mechanisms are used to block the security relevant actions that may cause security policy violations.

The rest of the paper is organized as follows. First, we survey on related work and the current state of the art (Section II). We then recall the Security-By-Contract model and put the base for a probabilistic extension of such a model (Section III). Afterwards, we introduce different formal definitions of prob-
ABSTRACT

Probabilistic compliance, involving app behavior, app contract, and security policy, are used to estimate the security guarantees associated to the app execution (Section IV). Then, we illustrate the architecture of the probabilistic Security-By-Contract model that has been implemented in an Android application (Section V). In particular, we show the monitoring and enforcement mechanisms implemented to ensure that the security policies are actually satisfied during the app execution. Then, we present a set of real-world experiments performed by using, as case studies, several mobile apps of the Android marketplace, both genuine and repackaged (Section VI). Finally, some conclusions terminate the paper (Section VII).

II. RELATED WORK

From the quantitative standpoint, the problem of finding an optimal control strategy is considered firstly in [2] in the context of software monitoring. In this setting, the system is represented as a Directed Acyclic Graph, while rewards and penalties with correcting actions are employed dynamically to find the optimal solution. Similarly, an encoding of access control mechanisms using Probabilistic Decision Processes is proposed in [3], where the optimal policy can be derived by solving the corresponding optimization problem. From a different perspective, Bielova and Massacci propose in [4] a notion of distance among traces representing the system behavior. If a trace is not secure, then it should be edited to a secure trace close to the non-secure one, where closeness is estimated in terms of the distance metric, thus characterizing an enforcement strategy. In [5], a system based upon system calls and Markov models is proposed to detect intrusions. This system analyzes the arguments of the system calls but is oblivious of the system call sequence. System call sequences and deterministic automata have been used in [6] to detect anomalies whenever the system call sequences differ from an execution trace known to be acceptable. This approach might suffer from high false alarm rate, since any trace different from a known one is considered as malicious. Our approach relaxes this condition, allowing the definition of more complex and flexible policies. Alterdroid [7] is a tool that compares the behavioral differences between an original app and an automatically generated version that contains modifications (faults) to detect hidden malware. The method of [8] proposes malware detection based on embeddings of function call graphs in a vector space capturing structural relationships. This representation is used to detect Android malware using machine learning techniques. The present work is not necessarily focused on malicious app, and potentially any kind of policy can be applied. Similarly, [9] classifies Android malware via dependency graphs by extracting a weighted contextual API dependency graph as program semantics to construct feature sets.

Referring to probabilistic models, probabilistic contracts have been firstly introduced in [10], where the contract generation is based on the analysis of the occurrences of system calls. The basic model relies on the Assume/ Guarantee paradigm for stochastic systems and the goal of verification is to analyze both reliability and availability aspects of such systems. The approach we propose is intended to extend the Security-by-Contract model in the probabilistic framework, in order to assess quantitatively the constraints behind the adoption of enforcement strategies at execution time.

III. FRAMEWORK MODEL

In this section, we present the logical model of the proposed framework, firstly introducing the security by contract approach, and then discussing a probabilistic extension to this known model.

A. Security By Contract

The Security-by-Contract (S×C) model [11] is based on three cornerstones: the application code A, the application contract C, and the client policy P. A contract is a formal, complete, and correct description of the security relevant behaviors of the application, like, e.g., security critical virtual machine API calls, or critical system calls. A policy is a formal, complete, and correct specification of the acceptable security-relevant behaviors that the application is allowed to execute on the device in which it is installed. Both contract and policy can be syntactically described by exploiting the same language.

The basic idea of a contract-based approach is the usage of the contract for guaranteeing that certain security conditions are satisfied. More in detail, using the contract, it is possible to check at deploy time, i.e., before the application execution, if the application satisfies the user policy or not. Let $\preceq$ denote the compliance relation between two of the previous elements. A contract-based approach guarantees that:

$$A \preceq C \preceq P \Rightarrow A \preceq P$$

Figure 1 reports the S×C components and workflow, which can be described as follows. The application comes with a contract describing the security relevant actions that the app could perform during the execution. As a first step, it is verified if the contract and the application are matched, i.e., if the contract is representative of the app behavior. This operation is named App Contract Matching and can be based on several methodologies spanning from proof-carrying code to trust in the application developer or on a certification
The policy requires the semantic redefinition of the two main introduction of probabilistic information in the contract and in execution traces is presented in [12].

A probabilistic contract is a document specifying all the security relevant actions that an application can perform together with the related execution probability. Thus, the probabilistic contract describes quantitatively the expected behavior for an app and represents a generalization of a standard contract. In fact, an action with null probability corresponds to an action that cannot be executed in the standard S×C model, whilst every action with a probability greater than 0 represents an enabled action for the standard S×C model.

With respect to the S×C model, a probabilistic contract cannot be built statically by examining, for example, the control flow graph of the application. The execution probabilities can either be manually estimated by the entity building the contract (certification authorities or the developer), or they can be generated dynamically, i.e., by observing the effective behavior of the app at run time. An automatic, tool-supported methodology to build probabilistic contracts from observed execution traces is presented in [12].

2) Matching Operations: On the architectural side, the introduction of probabilistic information in the contract and in the policy requires the semantic redefinition of the two main S×C control operations. Figure 2 depicts the architecture and workflow of the S×C×P model.

The Probabilistic App Contract Matching checks if the behavior of the application effectively matches the contract, by verifying that each specified action is executed with the probability expressed in the contract. Such a match is verified by taking into account tolerance thresholds, because when a probabilistic contract is built, the nondeterministic effect of behaviors that are not dependent from the app has to be considered. Notice that the Probabilistic App Contract Matching is satisfied by construction if the contract is generated dynamically through direct analysis of the app behavior.

The Probabilistic Contract Policy Matching verifies if the probability distribution expressed in the contract does not violate any condition specified in the policy, e.g., if no action is supposed to be more/less frequent than specific policy values. The requirements can be either strict, i.e., the probability values specified in the policies must be matched exactly, or can be approximated, by allowing a deviation depending on configurable thresholds. The rationale behind this relaxed compliance verification is again due to the unavoidable nondeterminism that affects the app execution.

IV. PROBABILISTIC COMPLIANCE

In this section, a model for the S×C×P framework is proposed with the aim of formalizing the notion of (approximated) probabilistic compliance and, as a consequence, extending in a quantitative setting the purely functional relation \(\preceq\).

As explained in the previous section, the application contract \(C\) is equipped with a model of the quantitative expected behavior of the application \(A\). Formally, such a model is given in the form of a theoretical estimated (sub-)probability distribution \(\pi\) associated to the domain of (relevant) actions \(Act\). More precisely, function \(\pi\), defined from a nonempty, at most countable set \(Act\) to \([0,1]\), is a discrete probability distribution over \(Act\) if: \(\sum_{a \in Act} \pi(a) = 1\). Intuitively, the contract \(C\) specifies that at each step of the application run the action \(a \in Act\) is chosen with probability \(\pi(a)\). Notice that, considering sub-probability distributions (such that, i.e., the summation above is \(\leq 1\)) permits to ignore irrelevant actions, which are excluded from \(Act\) but have non-zero probability.
Even in such a relaxed setting, the following theory can be applied with no restrictions.

On the other hand, the policy $P$ is described in terms of quantitative conditions over the execution frequency associated to the allowed security relevant actions. For instance, “the probability of action $a$ must be between 0 and 0.1” could be one such requirement, which, generally speaking, are defined to restrict the values that certain execution frequencies can assume at run time. Formally, the specification of $P$ is represented by a constraint satisfaction problem, see, e.g., [13] for the mathematical details.

In the following, we show how to employ the quantitative models surveyed above to relate application behavior, contract, and policy. Such relations are used to establish the probabilistic application compliance.

A. Estimating $A \preceq C$

Firstly, the compliance of the application with respect to the contract ($A \preceq C$ in the non-probabilistic case) cannot be checked statically through the application of some metric. In fact, we need to evaluate the quantitative run-time behavior of $A$. To this aim, notice that the sequence of actions observed during execution defines an actual discrete probability distribution. Such a distribution is expected to represent an experimental approximation of $\pi$. Hence, the aim is to compare, at each trial of the execution run, the distribution observed at run time against $\pi$, in order to verify whether the application behavior is compliant with the contract from a quantitative perspective. This similarity analysis must be conducted by taking into account a tolerance factor inversely proportional to the length of the execution run and, possibly, an estimated level of confidence associated to the result of the comparison.

In the following, we show how to apply classical statistical inference analysis to the estimation of such a compliance check directly at run time. The reader interested in the mathematical background can refer to [14], [15]. We first check directly at run time. The reader interested in the inference analysis to the estimation of such a compliance comparison.

Given a confidence level $l$, and the length $n$ of the execution run, can be expressed as: $E_n(a) = Z \cdot \sigma(a)$. Then, since the aim is to verify whether the actual estimate $\rho_n(a)$ measured at run time is compliant with the estimated probability $\pi(a)$, we evaluate the disequality:

$$|\rho_n(a) - \pi(a)| \leq E_n(a).$$

(3)

Definition 1. Given a confidence level $l$, we say that $A$ matches $C$ after $n$ execution steps (written $A \preceq_n C$) if for all $a \in Act$ it holds that (3) is satisfied.

The above definition can be extended to apply the compliance check at each step of an execution run of length $m$.

Definition 2. Given a confidence level $l$ and an execution run $\nu$ of length $m$, we say that $A$ matches $C$ on $
u$ if for all $1 \leq n \leq m$ it holds that $A \preceq_n C$.

Hence, contract compliance of the observed behavior of the application (represented by a sequence $\nu$ of actions) can be verified by checking whether $A$ matches $C$ on $\nu$.

With respect to the proposed approach, there is some analogy with formal methods applied to the comparison of the quantitative behavior of systems, like, e.g., in the formal setting of process algebra and behavioral equivalences [16]. The system behaviors associated to two probability distributions $\pi$ and $\rho_n$ can be described, e.g., by the probabilistic process algebraic terms:

$$S \triangleq \sum_{a \in Act} a[\pi(a)], S \text{ and } O_n \triangleq \sum_{a \in Act} a[\rho_n(a)], O_n$$

representing the theoretical estimated specification of the system and the experimental run-time observation, respectively. In both cases, the summations express the external choice among the several different events, which is governed by a probability distribution. The notation $a[p], P$ intuitively means that $a$ is executed with probability $p$ after which the process behaves as $P$. The semantics of the two process terms are given by probabilistic labeled transition systems with a unique state and a self-loop for each $a \in Act$ labeled with the action name $a$ and the probability $\pi(a)$ (resp., $\rho_n(a)$). These systems obey the generative model of probabilities and can
be compared with each other through an approximated variant of the classical bisimulation based equivalence check, which we refer as $\epsilon$-bisimulation. By following such an approach, see, e.g., [17]–[19], it turns out that the two systems behave approximately the same up to a tolerance threshold computed as $\epsilon = \max_{a \in \text{Act}}(|\pi(a) - \rho_n(a)|)$, similarly as demonstrated through the statistical inference check. However, differently from these approaches, which limit their contribution to the computation of such an $\epsilon$, in the setting of statistical inference analysis $\epsilon$ is given a reliability interpretation with respect to the desired confidence level. This interpretation is necessary to trade $n$ (the length of the experiment), the error estimation $\epsilon$, and the risk-related probability (provided by the confidence level) of guessing a (statistically significant) difference between the estimated behavior and the monitored behavior.

B. Estimating $C \preceq P$

The probabilistic compliance between contract $C$ and policy $P$ can be easily checked by observing that $\pi$ represents an evaluation for the constraint satisfaction problem modeling the policy. We first notice that, by definition, $\pi$ is complete as it includes all the variables of the constraint satisfaction problem. Hence, it is sufficient to check whether $\pi$ is also consistent, i.e., no constraints are violated by $\pi$. Indeed, we recall that an evaluation represents a solution for the problem if it is both consistent and complete [13].

**Definition 3.** We say that $C$ matches $P$ (written $C \preceq P$) if $\pi$ is a solution of the constraint satisfaction problem modeling $P$.

In order to relax the definition above, we add a tolerance threshold to the compliance check. Informally, $C$ approximates $P$ if $\pi$ is similar to a distribution that solves the constraint satisfaction problem modeling $P$. In analogy with the similarity checks of the previous section, we use the Chebyshev distance metric [20] to compare two distributions $\pi$ and $\pi'$:

$$d(\pi, \pi') = \max_{a \in \text{Act}}(|\pi(a) - \pi'(a)|)$$

which we use in the following approximated version of Def. 3.

**Definition 4.** We say that $C$ matches $P$ up to $\epsilon$ (written $C \preceq_\epsilon P$) if there exists a solution $\pi'$ of the constraint satisfaction problem modeling $P$ such that $d(\pi, \pi') \leq \epsilon$.

C. Estimating $A \preceq P$

Relation $\preceq_\epsilon$ can be applied also to relate $A$ and $P$, in which case the distribution to consider is $\rho_n$ instead of $\pi$. With this consideration in view, the following compliance result between $A$ and $P$ can be derived by combining linearly the two checks based on relations $\preceq_n$ and $\preceq$ previously discussed.

**Theorem 1.** If $A \preceq_n C$, with $\epsilon = \max_{a \in \text{Act}} E_n(a)$, and $C \preceq P$, then it holds that $A \preceq_{\epsilon + \epsilon} P$ with confidence level $l$.

The proof is a straightforward consequence of Defs. 1 and 3. The interpretation is as follows. Assume that the monitored behavior of the application statistically matches the contract with respect to the desired confidence level, and that the contract matches the quantitative requirements of the policy. Then, the application satisfies the policy with a margin dependent on the statistical error.

By replacing $C \preceq P$ by $C \preceq_{\epsilon} P$ we obtain an extension of the previous result showing that the level of approximation relating $A$ and $P$ is the sum of the margin errors introduced by each of the two compliance checks.

**Theorem 2.** If $A \preceq_{\epsilon_n} C$, with $\epsilon_1 = \max_{a \in \text{Act}} E_n(a)$, and $C \preceq_{\epsilon_2} P$, then it holds that $A \preceq_{\epsilon_1 + \epsilon_2} P$ with confidence level $l$.

D. Enforcing $A \preceq P$

It may be that the contract $C$ does not match the policy $P$ neither precisely nor approximately (see Defs. 3 and 4). Therefore, it would not be meaningful to check the compliance of the application $A$ with respect to the contract in order to estimate $A \preceq P$. Analogously, it is not possible to derive transitivity of the compliance of $A$ with respect to $P$ whenever $A$ does not match the contract (i.e., the hypothesis $A \preceq_n C$ of the previous theorems is not satisfied). In such cases, $A$ can be still executed and checked step-by-step against the policy $P$, possibly enabling the enforcement mechanism every time the application does not respect the conditions of the policy.

Formally, the compliance of $A$ with respect to $P$ is defined as a variant of Def. 1 in which we replace the distribution modeling the contract with a distribution solving the constraint satisfaction problem modeling $P$.

**Definition 5.** Given a confidence level $l$, we say that $A$ matches $P$ after $n$ execution steps (written $A \preceq_{l_n} P$) if there exists a solution $\pi$ of the constraint satisfaction problem modeling $P$ such that for all $a \in \text{Act}$ it holds that:

$$|\rho_n(a) - \pi(a)| \leq E_n(a).$$

We can argue similarly to define the corresponding variant of Def. 2. In practice, the check above is to be satisfied at each step $n$ of the execution run, otherwise the enforcement mechanism is activated.

V. ENFORCEMENT ARCHITECTURE

In this section, we describe the implemented architecture that can be exploited for both dynamic contract generation and policy enforcement. The proposed implementation has been designed for Android systems. Thus, the $S \times C \times P$ framework comes as an Android app, which is able to perform the operations previously discussed, directly on the device. Henceforth, the app will be referred as $S \times C \times P$-App.

The $S \times C \times P$-App is able to control actions performed by any other app on the device, together with a set of global actions performed by the operating system, such as controlling the light of the screen. The control of an action is performed by means of the hooking operation, i.e., putting a callback on the method invocation related to the action that has to be controlled. Once hooked, it is possible to control a method,
by performing actions before it is executed or immediately after. To this end, the proposed system exploits the Xposed-Framework [21], a toolkit for methods hooking available for Android. The Xposed Framework (or simply Xposed) is an advanced custom developer tool designed to give a much greater control on the Android system and on the running apps, compared to the one granted by the Android available APIs. The Xposed Framework can be installed on any Android device and release, however it requires the target device to be rooted (jailbroken), which may constitute a limitation for distribution. However, we argue that the S×C×P framework is not designed for, now, to be largely distributed among average users. Instead, it has to be considered as an advanced research tool for policy enforcement, which is ready to be integrated in specific environments where a single entity controls several devices (e.g., as in the case of a company that gives a business mobile phone to its employees).

The S×C×P app can be configured to work in two modes: (i) Logging Mode, which records every time an hooked action is invoked, and (ii) Enforcement Mode, which verifies if the hooked action is compliant with the policy when it is invoked, by stopping the behavior (enforcement-by-truncation) or modifying the outcome (enforcement-by-obligation) if the policy is not matched. Whenever a new app is installed on the device, the S×C×P-App intercepts the event and checks if a contract is available for it. The contract can be provided as an extension of the original AndroidManifest.xml file, or can be made available through an external repository. In the following, for the sake of simplicity we assume that the probabilistic App-Contract Matching is the only task that is not performed directly on the device at execution time. Such a validity check is, in fact, proven before the app execution on the mobile device. To this end, it is possible to assume that all the related operations, which must be conducted dynamically, are performed by running the app in a protected environment (such as a sandbox). As an alternative, contract and compliance check can be provided directly by the developer in such a way that the App-Contract Matching reduces to be an operation based on trust, using for example the same model proposed in [22].

The actions controlled by the S×C×P-App are:

- **Outgoing Sms Messages**: functions related to sending text messages, controlled to verify if the message is directed toward a number that is present in the contact list, or toward an unknown number.
- **Activity.onResume/Activity.onPause**: functions related to the execution of the application either in foreground or in background.
- **HTTP connections**: functions related to the opening of connections toward external servers.
- **User Present and Screen On/Off**: functions related to the interactions of the user with the device.

VI. EXPERIMENTAL RESULTS

Thanks to its expressiveness, S×C×P can be exploited to consider several aspects of an Android device life-cycle, depending on the specified policy. However, the proposed implementation is conceived for controlling security critical functionalities. Hence, it is applied in the present work to address security issues. In particular, the S×C×P framework has been used to tackle the actions of 270 malware samples divided into 8 malware families belonging to the Spyware and SMS-Trojan classes. These two malware classes include the most dangerous malware samples, since their malicious behavior directly affects both the user money and privacy. As a first step, the contract for each malware sample has been extracted dynamically, by executing the software in a protected sandbox, i.e., a controlled Android emulator, where the app could not cause any real damage. To this end, the logging mode of the S×C×P-App has been used, by collecting 10 traces of variable length for each sample, which are then used to generate the app contract. The used principle is that, with respect to the estimation of A ⊳ C (see Section IV-A), contract compliance of the observed app behavior is ensured with a 95% confidence level. Afterwards, by following Def. 3, the contract has been matched against a set of probabilistic policies with configurable probabilities to control malicious text messages and undesired data connections. The policies are as follows:

- **Policy 1** The app should not send text messages with a probability higher than \(x\%\), where \(x \in \{0, 10, 20, 30, 40, 50\}\).
- **Policy 2** The app should not perform http connections with a probability higher than \(x\%\), where \(x \in \{0, 10, 20, 30, 40, 50\}\).
- **Policy 3** The app should not send text message toward numbers that are not in the contact list with a probability higher than \(x\%\), where \(x \in \{0, 10, 20, 30, 40, 50\}\).
- **Policy 4** The app should not perform http connections toward numerical IP address (i.e., not toward a domain name) with a probability higher than \(x\%\), where \(x \in \{0, 10, 20, 30, 40, 50\}\).

Policies 1 and 3 are used to control outgoing text messages with different specifications on the recipient number. In particular, Policy 1 is designed to control those apps whose expected behavior is totally unrelated to text messages, i.e., the app is not supposed to send text messages to any contact. Thus, the action itself of sending a text message should be considered suspicious. On the other hand, Policy 3 only controls those messages sent to recipients whose number is not in the contact list. This policy is more suitable to control the behavior of apps which instead should interact with text messages, such as messaging apps, or SMS managers. We argue that in the normal usage of such a kind of app it is much more likely that a text message is sent to a known number, i.e., in the contact list, than to an unknown one. Still, through the usage of probabilistic policies it is possible to allow this unlikely operation, by tuning the relative probability execution.

Policies 2 and 4 are designed to control the outgoing data traffic, monitoring the connections opened by specific apps. Policy 2 is designed to control those apps that may generate
large amounts of traffic, or that, on the other hand, should not be supposed at all to perform network connections. Policy 4 is more specifically designed to control apps opening suspicious connections that are opened toward an IP address in the inet format, i.e., not toward a domain name. This kind of behavior is, in fact, typical of malware sending information to servers controlled by attackers (spyware). Thus, Policy 4 performs a finer-grained control on outgoing traffic, only filtering the suspicious one.

The results of the Contract-Policy matching against the 270 samples belonging to the 8 malware families are reported in Table I.

### TABLE I
**RESULTS OF CONTRACT-POLICY MATCHING FOR MALWARE FAMILIES.**

<table>
<thead>
<tr>
<th>Family Name</th>
<th>Class</th>
<th>Samples</th>
<th>Detected</th>
<th>Policy</th>
<th>Threshold</th>
<th>Trace Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>German</td>
<td>SMS-Trojan</td>
<td>25</td>
<td>Yes</td>
<td>1,3</td>
<td>20%</td>
<td>22</td>
</tr>
<tr>
<td>BaseBridge</td>
<td>SMS-Trojan</td>
<td>122</td>
<td>Yes</td>
<td>2</td>
<td>40%</td>
<td>86</td>
</tr>
<tr>
<td>DogOBar</td>
<td>SMS-Trojan</td>
<td>7</td>
<td>Yes</td>
<td>1,3</td>
<td>50%</td>
<td>20</td>
</tr>
<tr>
<td>Hippo</td>
<td>SMS-Trojan</td>
<td>15</td>
<td>Yes</td>
<td>1,3</td>
<td>10%</td>
<td>285</td>
</tr>
<tr>
<td>FakePlayer</td>
<td>SMS-Trojan</td>
<td>17</td>
<td>Yes</td>
<td>1,3</td>
<td>50%</td>
<td>17</td>
</tr>
<tr>
<td>Emni</td>
<td>Spyware</td>
<td>52</td>
<td>Yes</td>
<td>2,4</td>
<td>20%</td>
<td>18</td>
</tr>
<tr>
<td>TapSnake</td>
<td>Spyware</td>
<td>17</td>
<td>Yes</td>
<td>2,4</td>
<td>20%</td>
<td>15</td>
</tr>
<tr>
<td>OpFake</td>
<td>SMS-Trojan</td>
<td>14</td>
<td>Yes</td>
<td>1,3</td>
<td>30%</td>
<td>20</td>
</tr>
</tbody>
</table>

All malware samples have been detected as violating one or more of the four aforementioned security policies. The specific violated policy is reported in the Policy column. The severity of the violation has been estimated by tuning the policy configuration parameter. In fact, all the four policies have been tested by varying the value of the related probabilistic threshold. The Threshold column of Table I reports the highest (hence, less restrictive) value of the threshold \(x\) that causes the contract-policy mismatch.

Each SMS Trojan, as expected, violates both Policies 1 and 3. The device used for testing included real contacts in the contact list and for those apps showing functionalities to send text messages, at least one message has been sent during the test. Hence, sending text messages to unknown numbers (Policy 3) is an undesirable behavior.

Spyware apps, instead, have been found to violate Policies 2 and 4, except for Basebridge, which only violates Policy 2. This malware, in fact, directs the traffic toward a server with DNS. Still, in most of the samples Basebridge presents itself as an utility app similar to a memo, which should have no reasons to connect to the network, hence the misbehavior can be easily detected. The Trace Length column reports the average global number of critical operations collected per trace in order to build the sample contract. As shown, some apps have a limited trace length, in particular FakePlayer, Kmin and TapSnake. This is representative of the reduced interactions of some malicious apps, which do not trigger any of the critical actions monitored, except for those revealing the malicious behavior.

Table II reports the results for probabilistic contract-policy compliance performed, similarly as done for the previous experiment, on a set of genuine apps downloaded from Google Play, the official Android market. The contracts have been checked again on Policies 1 to 4. As shown, the majority of the tested apps have a contract that does not match Policy 2, related to data usage. The Threshold column of Table II reports the threshold value \(x\) of the most restrictive policy that is satisfied by the app contract. Therefore, it is interesting to notice the strong network interactions of the first three apps, which are supposed to be offline videogames, i.e., working even if the device is not connected to the network. In particular, the game CrossyRoad spends about 50% of the considered actions in performing network activities. Even if it is not by itself a security criticality, it is worth noticing how the \(S \times C \times P\) model can also be exploited to enforce usage policies that are able to improve the user experience, reducing the overhead generated by apps. Observing the Trace Length column, it is possible to notice the difference in the amount of actions with respect to the malware of Table I. Genuine and popular apps are highly interactive, especially for games. Still, controlling the number of actions is not enough to differentiate between malicious apps and genuine ones, since malicious code can also hide behind popular apps that are repackaged [12].

Finally, it is worth paying attention to the results for the Whatsapp application. This app has been tested against Policies 1 and 3 since, being a communication app exploiting the Internet, it would be pointless to test it against Policy 2. Whatsapp sends a single text message at the very first startup to register the user; then, the overall relative probability of such a security relevant event is less than 10%. On one hand, at the beginning such an event is tolerated by the margin errors ensured by the statistical inference check at the base of the app-contract matching (see Section IV-A). On the other hand, its overall low frequency allows the app to remain compliant with respect to any of the considered probabilistic policies, except for the case in which no SMS is allowed at all (i.e., \(x = 0\)). In a purely functional setting, it is worth noticing that Whatsapp does not pass the compliance test of the standard \(S \times C\) control if outgoing SMS messages are not allowed by the

### TABLE II
**CONTRACT-POLICY MATCHING FOR GENUINE APPS.**

<table>
<thead>
<tr>
<th>App Name</th>
<th>Type</th>
<th>Policy</th>
<th>Threshold</th>
<th>Trace Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow</td>
<td>Game</td>
<td>2</td>
<td>20%</td>
<td>158</td>
</tr>
<tr>
<td>AngryBirds</td>
<td>Game</td>
<td>2</td>
<td>30%</td>
<td>322</td>
</tr>
<tr>
<td>CrossyRoad</td>
<td>Game</td>
<td>2</td>
<td>50%</td>
<td>251</td>
</tr>
<tr>
<td>FruitNinja</td>
<td>Game</td>
<td>2</td>
<td>10%</td>
<td>537</td>
</tr>
<tr>
<td>KingCalculator</td>
<td>Calculator</td>
<td>2</td>
<td>&lt; 10%</td>
<td>144</td>
</tr>
<tr>
<td>WhatsApp</td>
<td>Communication</td>
<td>1,3</td>
<td>&lt; 10%</td>
<td>242</td>
</tr>
</tbody>
</table>

### TABLE III
**ENFORCEMENT RESULTS.**

<table>
<thead>
<tr>
<th>App Name</th>
<th>Type</th>
<th>Trace Length</th>
<th>C.I.</th>
<th>Data Policy (1,4)</th>
<th>SMS Policy (1,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AngryBirds</td>
<td>Genuine</td>
<td>30</td>
<td>95%</td>
<td>Enforced (10%)</td>
<td>Not Enforced</td>
</tr>
<tr>
<td>CrossyRoad</td>
<td>Genuine</td>
<td>20</td>
<td>95%</td>
<td>Enforced (40%)</td>
<td>Not Enforced</td>
</tr>
<tr>
<td>FruitNinja</td>
<td>Genuine</td>
<td>30</td>
<td>95%</td>
<td>Enforced (10%)</td>
<td>Not Enforced</td>
</tr>
<tr>
<td>KingCalculator</td>
<td>Genuine</td>
<td>10</td>
<td>95%</td>
<td>Enforced (0%)</td>
<td>Not Enforced</td>
</tr>
<tr>
<td>Flow</td>
<td>Genuine</td>
<td>30</td>
<td>95%</td>
<td>Enforced (20%)</td>
<td>Not Enforced</td>
</tr>
<tr>
<td>fake</td>
<td>Malicious</td>
<td>2</td>
<td>95%</td>
<td>Not Enforced</td>
<td>Enforced (99%)</td>
</tr>
<tr>
<td>DogOBar</td>
<td>Malicious</td>
<td>2</td>
<td>95%</td>
<td>Not Enforced</td>
<td>Enforced (99%)</td>
</tr>
<tr>
<td>BaseBridge</td>
<td>Malicious</td>
<td>2</td>
<td>95%</td>
<td>Not Enforced</td>
<td>Not Enforced</td>
</tr>
<tr>
<td>Hippo</td>
<td>Malicious</td>
<td>17</td>
<td>95%</td>
<td>Not Enforced</td>
<td>Not Enforced</td>
</tr>
<tr>
<td>FakePlayer</td>
<td>Malicious</td>
<td>17</td>
<td>95%</td>
<td>Not Enforced</td>
<td>Not Enforced</td>
</tr>
</tbody>
</table>
policy. On the other hand, the S×C check is not meaningful at all if outgoing messages are allowed by the policy.

The apps that are not compliant with a policy can be either removed or executed under the control of the enforcer described in the previous section. Table III reports the results concerning the enforcement of policies on a subset of monitored apps, by applying the method of Section IV-D (app-policy matching is based on 95% confidence level).

The columns on Data and SMS policies report (i) whether the policy has been enforced and, if enforced, (ii) the maximum value of the probability threshold $x$ causing the enforcement of the policy. The Trace Length column reports the number of relevant actions executed by the app before the enforcement of the policy. Hence, it is worth noticing that the Trace Length value for all malicious apps is equal to 2, meaning that the behavior violating the policy is effective in the very first steps performed by the app. Furthermore, we point out that, enforcing the policies on genuine apps does not negatively modify the experience of the user, still preserving all the desired app functionalities. The reason is that several genuine apps mainly use http connections to provide in-app advertisement, which is not effectively necessary to the correct execution of the app, providing instead an undesired overhead for the user. Hence, in such cases, the policy enforcement represents a way to keep under control such an overhead.

VII. CONCLUSION

The quantitative approach to the application of the Security-by-Contract framework that we presented in this paper has manifold advantages. On one hand, it allows to establish not only the nature of the security relevant activities that have to be monitored to ensure the security policy, but also the expected frequency with which such activities can be observed. In fact, for many cases, such as an outgoing SMS, the execution frequency may affect the validation of the security policy. On the other hand, the use of quantitative information allows to introduce approximations in the specification of behavior and contract of real-world, complex apps. For these types of apps, precise evaluations and exact matching could be hard to achieve and, even more important, could lead to misleading results. As a side effect of the use of quantitative approaches, we also observed that it is possible to trade the efficiency of the securing mechanisms and the user quality of experience.

Currently, the quantitative evaluation is based on the frequency of security relevant actions. As future work, we plan to express the contract/policy in terms of a probabilistic labeled transition system specifying the execution traces that can be observed at run time, together with their execution probability, similarly as done, e.g., in [23]. Then, the probabilistic compliance will be based not only on the action execution frequency, but also on the matching of the observed behavior with respect to such a structure.

REFERENCES