Probabilistic Contract Compliance for Mobile Applications

Gianluca Dini, Fabio Martinelli, Andrea Saracino, Daniele Sgandurra

Istituto di Informatica e Telematica, CNR, Italy
daniele.sgandurra@iit.cnr.it

IWSMA 2013
05/09/2013
Outline

1 Mobile Security

2 PICARD
   - Contract
   - ActionNode
   - Contract Generation
   - Contract Compliance

3 Results
   - Tests
   - Conclusion
Part I

Mobile Security
Current operative systems for mobile devices are based upon the concept of apps.

Distributed through on-line marketplaces, such as the AppStore, Google Play: new paradigm for application distribution.
**Apps**

- **Users** browse apps on markets and install them on their devices.
- This model is affected by some **security** and **trust** issues:
  - some apps are produced by **third-party developers**, which can be malicious ones,
  - some markets do not ensure the **security** and **quality** of the published apps.
Repackaged Application

- A large number of repackaged applications has been found hidden in apps distributed in the Android markets.
- These are apps that look and work as genuine applications, but they hide inside new code that misbehaves in the background:
  - e.g. by contacting fake advertisement sites.
A special class of repackaged applications is that of Trojanized applications, which are apps that have malware inserted into them. Trojanized applications form more than 80% of the total malware. Detection is hard: observable behavior of a good application is mostly the same as of a Trojanized/repackaged version.
What We Propose

- **PICARD** (Probabilistic Contract on AndRoiD), a framework aimed at detecting repackaged apps based upon probabilistic contracts.
- **Contracts** describe the sequences of actions that an application is allowed to perform when running.
- At run-time, PICARD monitoring system verifies the compliance of the app trace with its contract using probabilistic contract matching.
Part II

PICARD
A **contract** is a document that describes the expected behavior of an application.

Possible representations:

- Natural Language.
- Formal Specification Language.
- **Execution Graphs.**
- ...
Contract in Picard

- A contract is defined by monitoring some executions of the program to extract its behavior.
- A contract clusters the possible actions of an application into two sets: likely actions and unlikely actions:
  - at run-time, an unlikely action that is performed several times should be considered a misbehavior.
High Level View of Contract Generation

Steps performed by PICARD to build an application contract:

1. **System calls** issued by an application execution during an offline-training phase are used to build a **trace**.

2. Multiple traces are merged to build a **multi-graph of system calls**.

3. PICARD defines an algorithm to find **relations** among consecutive system calls in the multi-graph:
   - creates a **multi-graph of ActionNodes**, i.e. clusters of syscalls.

4. From this multi-graph, PICARD extracts a **probabilistic automaton**, which represents the contract.
Application Trace

- In PICARD, an application trace is a sequence of system calls.
- PICARD expresses the trace through an oriented graph $G = (V, E)$, where each system call is a node $v_i \in V$, and edges $e_{i,j} \in E$ represent the transition from a system call $v_i$ into the next one $v_j$.
- The resulting graph may contain more occurrences of the same edge (multi-graph).
Merging Multi-Graphs

- Several execution traces (i.e. multi-graphs) are merged.
- From the resulting multi-graph, firstly the multi-graph of ActionNodes is generated, then the probabilistic automaton is computed.
- To this end, PICARD introduces the concept of node clustering (i.e., ActionNodes) and applies elements of probabilistic contract theory.
To represent contracts, we introduce the notion of ActionNode:

- cluster of related system calls: sub-graph of system calls that are consecutively issued in the trace and that are bound by some relation;
- form an action (a high-level operation).

The relation should produce a meaningful action:

- an ActionNode is composed by the automaton of the system calls performed consecutively on the same file descriptor.
ActionNode

- An application trace can be seen as a graph whose nodes are actions.
- The framework can define a larger number of nodes, hence more detailed signatures.
- The nodes that compose an ActionNode, i.e. system calls, are called SysCallNodes.
- We consider as SysCallNodes system calls that act on files, namely the open, read, write, close, ioctl.
ActionNode Examples
Algorithm To Generate ActionNodes

- A new ActionNode is created each time a system call is issued with an argument different from the one of the former issued system call.

- If the system call is the same as the previous one, with the same argument, then an edge is generated that goes back on the same SysCallNode.

- If the argument is the same, but the system call is different, a new SysCallNode is added to the current ActionNode, provided it has not been already created:
  - an arc is added from the current SysCallNode to this new SysCallNode.
Pseudo-Code For Generating ActionNodes

```plaintext
for each SysCall s in Multigraph:
    updateGraph(s);

updateAutoma(SysCall currentSysCall)
{
    if (currentActionNodeGraph == null)
        currentActionNodeGraph = new ActionNodeGraph();

    if (currentSysCall.filename != currentActionNodeGraph.filename)
    {
        ActionNode newActionNode = new ActionNode(←
            currentActionNodeGraph);
        automa.update(newActionNode);
    }
    else
    {
        currentActionNodeGraph.update(currentSysCall)
    }
}
```
New SysCallNode

New syscall, same parameter
New ActionNode

Syscall with new parameter
Contract Generation

- Each ActionNode is then converted in an automaton that represents the high-level operation.
- Then, a multi-graph of ActionNodes is generated.
- In the next step, PICARD converts the multi-graph of ActionNodes into a probabilistic automaton.
- The automaton contains the same nodes of the multi-graph, but multiplicities of edges are not taken into account.
- On the other hand, multiplicities of edges are used to compute the related probabilities in $A$. 


This transformation induces a probability distribution for each node.

Each edge 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 \( v_i \).

It is worth noticing that this automaton can be described by means of a Markov-Chain, whose states are the ActionNodes of the application, and the transition matrix reports the probabilities of the automaton edges.

This probabilistic automaton is used to define the probabilistic contract of the application behavior.
Multi-graph Transformation
Contract Compliance

- PICARD incrementally builds the monitored application behavior in the same way it has built the contract.
- The run-time behavior is represented through a trace of ActionNodes.
- Each ActionNode is a state in a Markov Chain transition matrix, and \( m \) is the number of ActionNodes in the contract.
- The probability of being in each state \( i \) at the step \( n \) is computed as the \( i \)-th element of the vector \( p^{(n)} = p^{(0)}P^n \), where \( p^{(0)} \) is the initial state vector.
- By matching the trace of ActionNodes with the contract, PICARD detects a misbehavior when an application is in an ActionNode of the contract automaton that is strongly unlikely, or the ActionNode, or edge, does not exist.
PICARD defines three levels of contract violation:

1. **Probability Misbehavior**: the probability of being in the current state is very low but not null.
   - This means that the application is performing several times an action that is strongly unlikely.
   - Given a threshold $0 < \theta < 1$, if $p^n_i < \theta$, then a misbehavior is detected.
   - The value of $\theta$ is function of the length of the considered subtrace and the number of states in the Markov chain.
2 Missing-Edge Misbehavior: the probability of being in the current state \( i \) is zero, because the edge from \( j \) to \( i \) does not exist in the contract, where \( j \) is the state at the step \( n-1 \).

- This is the case of a mimicry attack.
- The application performs normal high-level operations, not considered dangerous as standalone, but that may harm the device if performed in a specific malicious sequence.
3 Missing-ActionNode Misbehavior: the probability of being in the current state $i$ is zero, because the current state (ActionNode) does not exist in the contract.

- An unknown high-level operation has been performed.
- This is the strongest misbehavior.
- In the case of a repackaged application, this ActionNode constitutes a high-level operation such as sending an SMS, or accessing to a system file, which is never performed on the original version.
Contract Compliance

At run-time, when an ActionNode is created, PICARD applies these steps:

1. it checks if the ActionNode exists in the contract: otherwise, PICARD assumes it is a violation of **Missing-ActionNode Misbehavior** type;

2. it verifies if an edge from the previous ActionNode to the current one exists in the contract; otherwise, PICARD assumes it is a violation of **Missing-Edge Misbehavior** type;

3. if the two former conditions are met, PICARD computes the probability of being in the current state, according to the contract. If \( p^n_i < \theta \), PICARD assumes it is a violation of **Probability Misbehavior** type.
**Contract Compliance**

- PICARD resets the trace of system calls and updates the trace of ActionNodes executed with the current ActionNode.
- If one of the three misbehaviors described formerly is detected, the application is not considered compliant with the contract.
Part III

Results
To test the effectiveness of PICARD, we have built a prototype to:

1. define the contracts from the application traces,
2. check the compliance of the applications run-time behavior with respect to their contract.

We have analyzed two case studies:

1. proof-of-concept of repackaged applications, by updating all the sample applications provided with the Android SDK;
2. real applications and their repackaged version, found on official market and on malware archive site.
Example of Repackaged Applications

- **Tic Tac Toe** sample application provided with the Android SDK.
- We modified the application code to send an SMS message each time the user opens the menu to change the game skin.
- The SMS is sent stealthily, since Android does not advise the user when an SMS is sent by an application.
- Moreover, the messages sent by an application are not stored in the message outbox.
## Multi-graph Transformation

### (a) Original Application

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>0.1308</td>
<td>0.3349</td>
<td>0.5312</td>
<td>0.0031</td>
<td>0.1661</td>
<td>0.0380</td>
<td>0.2792</td>
<td>0.1313</td>
<td>0.0028</td>
<td>0.0346</td>
<td>0.0346</td>
<td>0.4813</td>
<td>0.4813</td>
</tr>
</tbody>
</table>

### (b) Repackaged Application

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>0.5</td>
<td>0.4475</td>
<td>0.6292</td>
<td>0.0043</td>
<td>0.2461</td>
<td>0.0380</td>
<td>0.1313</td>
<td>0.0028</td>
<td>0.0346</td>
<td>0.0346</td>
<td>0.4813</td>
<td>0.4813</td>
<td>0.2075</td>
</tr>
</tbody>
</table>

---

*34/41 Probabilistic Contract Compliance for Mobile Applications*
Repackaged Applications

- The **automaton** of the malicious Tic Tac Toe is more complex than the original one.
- E.g., in the malicious trace, several **new high-level operations** have been detected: an example of missing ActionNode is node 11.

![Diagram of automaton]
Example of Trojanized Applications

- We report the results on the game **Baseball Superstars 2010**, which is a complex baseball simulation game.
- We found a repackaged version of the same game, which has a user-observable behavior identical to the original one.
- However, the repackaged version is infected by a Trojan malware called **Geinimi**.
- The malware steals private user data, like the IMEI code, and tries to send them to an attacker via **SMS** or Internet.
Trojanized Applications

- Several traces have been collected while playing the game, exploring the various play modes.
- The probabilistic automaton extracted from these executions graph has 13 states.
- Higher complexity of a real application with respect to the proof of concept formerly analyzed.
- Detecting non-compliant behavior may result more difficult with the increase of the application complexity.
Contract for the Game Baseball Superstars 2010
Trojanized Applications

- The traces of the repackaged application resulted in the generation of four ActionNodes that are not contained in the contract.
- These misbehaviors are related to the various attempt to access the IMEI code.
- Due to the insurgence of this unknown, malicious, operations, the application is not considered compliant with the contract and is recognized as repackaged.
- PICARD proved to be effective also in this case by correctly classifying both several traces of the genuine application and of the repackaged one.
Conclusion and Future Works

- We have presented **PICARD**, a framework to detect repackaged applications on Android systems through probabilistic contracts.
- The contract is represented using a **clustered multi-graph**, where each node represents a specific high-level operation, which is converted into a **probabilistic automaton**.
- At run-time, the application behavior is checked against the contract and a **misbehavior** is detected when the application is in a state of the automaton that has a low or null probability according to the probabilistic contract.
- We are planning to extend this work by **increasing the number of monitored system calls** and using **new relation among system calls**.
Questions?

- E-mail: daniele.sgandurra@iit.cnr.it