Runtime Verification For Android Security

Richard Allen
Swansea University, UK

Viva 13 December 2021

Monitoring For Collusion

Why Is Android Security An Important Topic?

- In 2019 there is reported 3.4 billion smart phone users world wide[1], 1.6 billion are Android devices[2].
- Attacks happen to such extent that the McAfee Q1 2020 threat reports opens with the headline "Mobile Malware Is Playing Hide and Steal"[3].
- We are monitoring for security on one concrete example: app collusion for information theft.

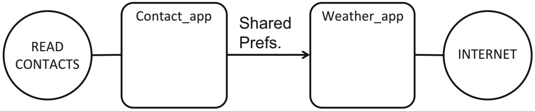
[1] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] https://www.statista.com/statistics/543185/worldwide-internet-connected-operating-system-population/

[3] https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf

Collusion: Specific Attack On Android

By collusion we mean where two or more applications communicate between themselves in order to use their combined permissions to perform malicious activities like stealing information.
This was theorised at least as early as 2011 and found to be occurring in 2017 [4].

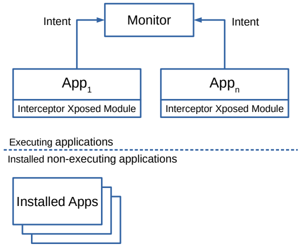


[4] J. Blasco, T. Chen, I. Muttik, and M. Roggenbach. Detection of app collusion potential using logic programming. Journal of Network and Computer Applications, 105, 06 2017

Intercepting Operating System Calls in Android

Xposed Framework[5]:

- Modifies the Linux Zygote process from which all processes are forked.
- This allows us to intercept operating system calls made by all processes.


Intent
Monitor
Intent
App$_1$
Interceptor Xposed Module
App$_n$
Interceptor Xposed Module
Executing applications
Installed non-executing applications
Installed Apps

Our Implemented Monitoring Architecture

[5]First released in 2012 by rovo89

Monitoring With the Rosu-Havelund Algorithm

Rosu-Havelund Algorithm

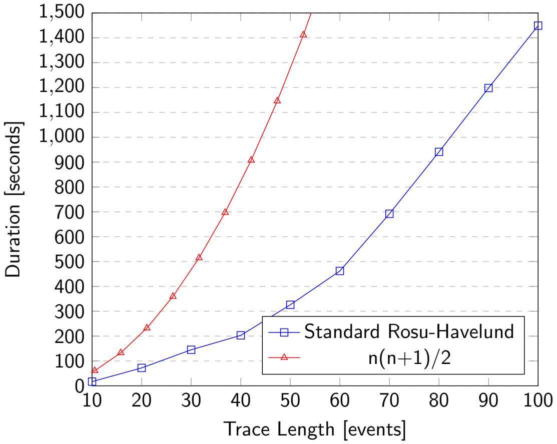
We attempted to use the Rosu-Havelund Algorithm [6] as an alternative to Buchi automata to evaluate LTL over a trace.

- Uses logic of the ‘future’, i.e, Always $\Box$, Eventually $\diamond$, Next $\circ$, Until U.
- Collusion For Information Theft between two apps: $\diamond(q \wedge (\diamond s \wedge (\diamond r \wedge (\diamond p))))$.
- Traverses the trace from the latest event to the earliest to implement future operators.
- Each time an event occurs the trace grows and the computational steps to evaluate the trace increases.
- We theorise the number of computational steps follows n(n+1)/2, therefore the complexity is $O(n^2 * |\varphi|)$ where n is the number of events in the trace and φ is the size of the formula.

[6] Grigore Rosu and Klaus Havelund. Synthesizing Dynamic Programming Algorithms from Linear Temporal Logic Formulae. RIACS, 2001

Complexity Analysis

Given the collusion formula we measured:



This makes the algorithm unsuitable for realtime monitoring.

Monitoring With the Reverse Rosu-Havelund Algorithm

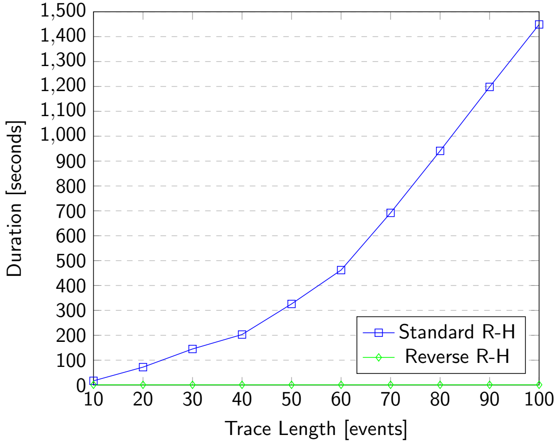
Reverse Rosu-Havelund Algorithm

- Traverses the traces in the opposite direction to standard R/H algorithm.
- Reduces complexity in realtime monitoring by reusing the result of the previous evaluation, therefore only the latest event has to be evaluated.
- Comes at the expense that the LTL operators that deal with future events (next, eventually) cannot be implemented because we do not know what future events will be. Operators that are concerned with past events are implemented (previous, once) instead.
- Collusion property expressed with past operators:

$$\diamondminus(p \wedge \diamondminus(r \wedge \diamondminus(s \wedge \diamondminus q)))$$

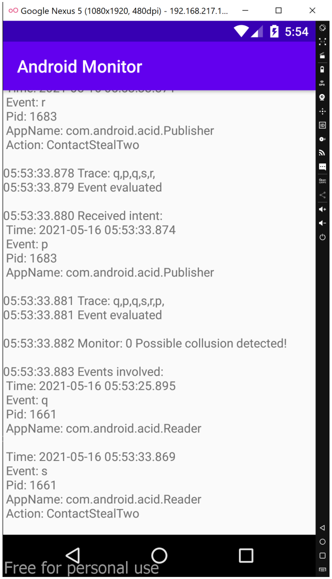
Complexity

Given the collusion formula we measured:



Suitable for realtime monitoring: takes in the order of ≤10ms per new event.

Live Demo



```
 Event: r
 Pid: 1683
 AppName: com.android.acid.Publisher
 Action: ContactStealTwo

05:53:33.878 Trace: q,p,q,s,r,
05:53:33.879 Event evaluated

05:53:33.880 Received intent:
 Time: 2021-05-16 05:53:33.874
 Event: p
 Pid: 1683
 AppName: com.android.acid.Publisher

05:53:33.881 Trace: q,p,q,s,r,p,
05:53:33.881 Event evaluated

05:53:33.882 Monitor: 0 Possible collusion detected!

05:53:33.883 Events involved:
 Time: 2021-05-16 05:53:25.895
 Event: q
 Pid: 1661
 AppName: com.android.acid.Reader

 Time: 2021-05-16 05:53:33.869
 Event: s
 Pid: 1661
 AppName: com.android.acid.Reader
 Action: ContactStealTwo
```

Runtime Monitoring For Security (Collusion) In Context

- Model checking apps has false positives (over approximation), works for small apps only[7].
- Static analysis has false positives (no reachability analysis)[8].
- Machine learning has false positive and false negatives[9].

Runtime monitoring is effective!

Based on the positive results of my project, Swansea University and Coventry University are applying for funding for monitoring for security in an Innovate UK call.

[7] Software Model Checking for Mobile Security(...). Irina Mariuca Asavoae, Hoang Nga Nguyen, Markus Roggenbach. SPIN 2018: 3-25

[8] Detection of app collusion potential using logic programming. Jorge Blasco, Thomas M. Chen, Igor Muttik, Markus Roggenbach. J. Netw. Comput. Appli. 105: 88-104 (2018)

[9] Towards a threat assessment framework for apps collusion. Kalutarage, Harsha Kumara and Nguyen, Hoang Nga and Shaikh, Siraj Ahmed. Telecommunication Systems. Springer US 2017: 1-14