Android Collusion Detection Using Linear Temporal Logic

Richard Allen, Markus Roggenbach
Swansea University, UK

MT-CPS 2021

Contents

Monitoring For Collusion

Why Is Android Security An Important Topic?

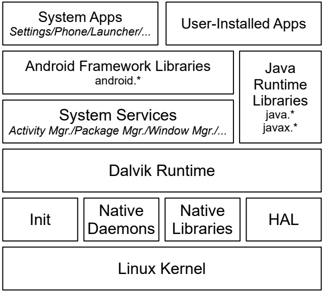
- In 2019 there is reported 3.4 billion smart phone users world wide[1], 1.6 billion are Android devices[2].
- Attacks happen to such extent that the McAfee Q1 2020 threat reports opens with the headline "Mobile Malware Is Playing Hide and Steal"[3].
- We are monitoring for security on one concrete example: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

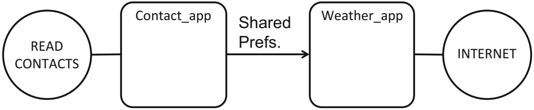
What Is the Android Security Concept?



- Linux Kernel separates processes into sandboxes where they are unable to reference each others address space directly.
- Android restricts access to sensitive resources to only those applications that have been granted permission by the user.
- Applications can communicate with each other to enhance features but this can be exploited to share permissions.

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

App Collusion and Operating System Calls

General Definition[5]: There is a non-singleton set S of apps performing a threat such that:

- Each app in S contributes the execution of at least one action to the threat.
- Each app in S communicates with at least one other app.

Observation:

- Accessing protected resources in Android (actions) and
- communication

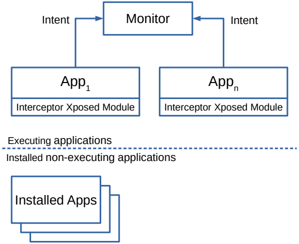
require operating system calls.

[5] Asavoae et al: Detecting Malicious Collusion Between Mobile Software. Applications: The Android Case. Data Analytics and Decision Support for Cybersecurity, Springer 2017

Intercept Operating System Calls in Android

Xposed Framework[6]:

- 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

[6] First released in 2012 by rovo89

Monitoring With the Rosu-Havelund Algorithm

Rosu-Havelund Algorithm

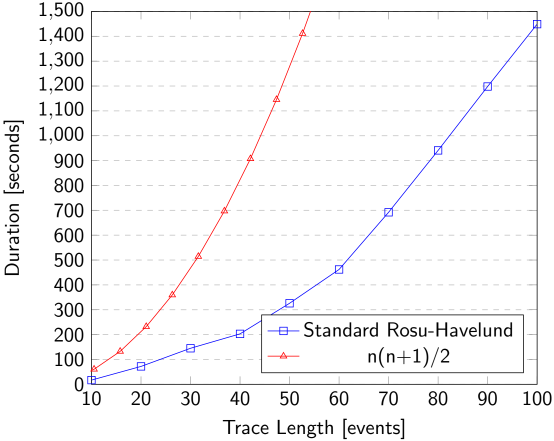
We attempted to use the Rosu-Havelund Algorithm [7] as an alternative to Buchi automata to evaluate LTL over a trace.

- 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 $O(n(n+1)/2 * |\varphi|)$ where n is the number of events in the trace and φ is the size of the formula.

[7] 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).

Collusion Property in LTL with the Once operator

The Once, $\diamondminus$, operator described by Pnueli[8] holds if an event has occurred in the present or at some preceding time.

- Once semantics:
  $(\sigma, j) \models \diamondminus p \textit{ iff } (\sigma, k) \models p \textit{ for some } k,\ 0 \leq k \leq j.$
- For comparison the eventually semantics:
  $(\sigma, j) \models \diamond p \textit{ iff } (\sigma, k) \models p \textit{ for some } k \geq j.$

[8] Zohar Manna, Amir Pnueli. The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag 1992

- Our collusion property looks for a publish event, preceded by a receive event, preceded by a send event, preceded by a query event.

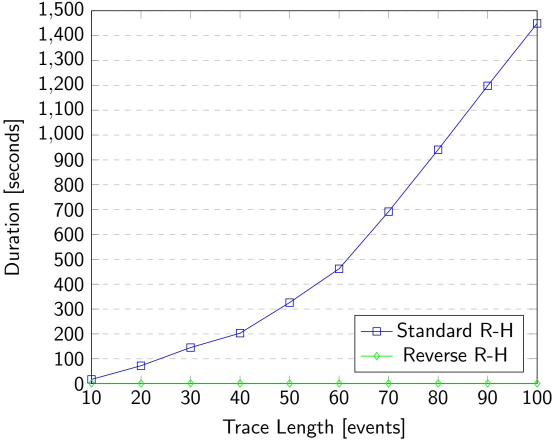
$$\diamondminus(p \wedge \diamondminus(r \wedge \diamondminus(s \wedge \diamondminus q)))$$

Conjecture

- Security properties can always be described in terms of past events because we are looking for when a security policy *has* been broken.
- For instance, a policy might be that there should be no audio taken from the microphone if there are no calls in progress.
- In that case we would look for an audio read event to have taken place with no preceding call events.
- Pneuli describes the past temporal operators as ‘a symmetric counterpart to each of the future operators. While the future formula describes a property holding at a suffix of the model (...), a past formula describes a property of a prefix of the model’

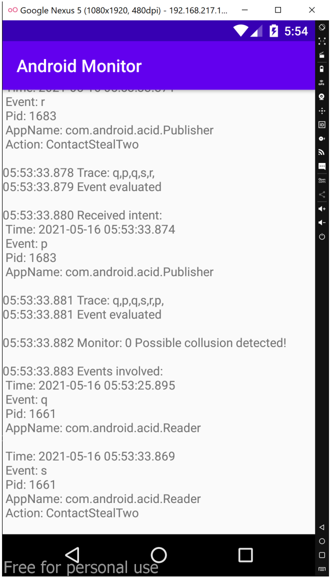
Complexity

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 In Context

- Runtime monitoring has false positives (no checks for metadata), is effective.
- Model checking apps has false positives (over approximation), works for small apps only[9].
- Static analysis has false positives (no reachability analysis)[10].
- Machine learning has false positive and false negatives[11].

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

[10] 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)

[11] 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

Conclusion

Summary

- Monitoring for Android security is possible.
- Requires "logic of the past".
- It is useful to monitor on abstract events.

It is possible to monitor for collusion without false positives if we restrict collusion to attempts within a limited timeframe.

Future Work

- Systematic experimentation with the monitors.
- Monitor for more properties, e.g. collusion over longer chains (not only 2 apps).
- Investigate if deriving monitors for concrete events from monitors for abstract events leads to better results than using monitors derived from LTLFO.
- Cover more Android versions, integrate Xposed framework into monitor, provide a security monitoring app.