Using Authorization Logic to Capture User Policies in Mobile Ecosystems

Joseph Hallett
University of Edinburgh
joseph.hallett@sms.ed.ac.uk

David Aspinall
University of Edinburgh
david.aspinall@ed.ac.uk

1. INTRODUCTION

Mobile devices let users choose the software they want to run. App stores offer a wide range of apps for users to pick from. Users make the decision of which apps to use based on their own policies. Sometimes this is based on what the store shows them first [6], other times it is based on information about the app—ratings and the permissions it requests [3].

There is some evidence that users tend to fall into patterns when picking apps [4]. By capturing these policies explicitly they can be enforced more consistently by a machine, lessening the burden on users to decide which apps they want. Furthermore whilst some expert users could write their own policies formally, others just want help. By sharing policies users can gain the benefits of expert written security policies without having to become experts themselves

2. POLICIES AND APPPAL

2.1 AppPAL

To write policies we have used the AppPAL authorization logic. AppPAL is a language, based on SecPAL [2], that can describe what it means for an app to be installable. AppPAL requires all statements be made by an explicit speaker, which enables rich delegation relationships to be expressed in policies. AppPAL is implemented as a Java library and runs on the Java and Dalvik virtual machines.

Alice, a user, can state that a specific app is installable.

```
"alice" says "com.rovio.angrybirds" isInstallable.
```

Alternately she can specify that an app is only installable if a set of constraints (facts checkable by inspecting the app directly, or by static analysis) are met. She can also delegate parts of the decision to third parties, who in turn must state facts about an app for it to be accepted. An example might be a that she will only install apps without the `LOCATION` permission that meet her *not-malware* policy (which she trusts McAfee to decide for her).

```
"alice" says App isInstallable
  if "not-malware-policy" isMetBy(App)
  where hasPermission(App, "LOCATION") = False.

"alice" says "mcafee" can-say 0
  "not-malware-policy" isMetBy(App).
```

2.2 Policies

Lin et. al. showed that users fall into four behaviour patterns when picking apps [4]. Conservative users felt uncomfortable allowing an app access to any personal data or functionality for any reason; unconcerned users felt okay allowing access to any data for almost any reason. Advanced users were more picky, for example comfortable allowing apps access to location data for core functionality but not for advertising purposes; whereas fence sitters (the largest cluster) varied about what they found uncomfortable but were broadly against collection of personal data for advertising purposes.

Using these behaviors as a start point we wrote the following AppPAL policies to describe their behaviour, each of increasing strictness, each banning increasing numbers of permissions.

	conservative	advanced	fencesitter	unconcerned
GET_ACCOUNTS	✗	✗	✗	✗
ACCESS_FINE_LOCATION	✗	✗	✗	
READ_CONTACT	✗	✗	✗	
READ_PHONE_STATE	✗	✗		
SEND_SMS	✗	✗		
ACCESS_COARSE_LOCATION	✗			

McAfee classify malware into several categories. The *malicious*, and *trojan* categories describe traditional malware, but there are other categories for PUS (potentially unwanted software). Using AppPAL we can write policies to differentiate between these kinds of malware. This allows us to explore the difference between users with dangerous apps installed and those who install merely unsavoury apps.

```
User says McAfee can-say "malware" isKindOf(App).
McAfee says "trojan" can-act-as "malware".
McAfee says "pus" can-act-as "malware".
```

3. EXPERIMENTS

3.1 Aims

We have two aims:to introduce AppPAL as a language for modelling and describing app installation policies, and to measure the extent users currently enforce app installation policies.

Our assumptions include:

- Downloaded apps used for experiments are not the same version as those used by users; and may have different permissions. Permissions tend to grow in apps over time [7] (in part due to increased system per-

missions) so we might find the apps allowed are more permissive now than when the user installed them.

- We do not have the full user purchase history. We can only test policies on the apps we know about. Consequently a user may have apps installed that break the policy, but which we do not know. This is a limitation of the Carat data set.

3.2 Methodology

We want to test how well these policies capture user behavior. The Carat project [5] have a large, partially anonymized[1], database of app installations by users. By calculating the hash of a package name you can see which users installed which apps.

From an initial database of over 90,000 apps, and 55,000 users who on average installed just under 90 apps each, we unmasked 4,300 apps (on average half the users installs). Disregarding system apps (such as `com.android.vending`) and very common apps such as Facebook, Dropbox, Whatsapp, and Twitter we further reduced the set to an average of twenty apps per user that they installed.

Using this data, and the apps themselves collected from the Google Play Store and Android Observatory [1] we checked which apps satisfied which AppPAL policies. From here we created a table of whether a user's apps satisfied a policy.

3.3 Results

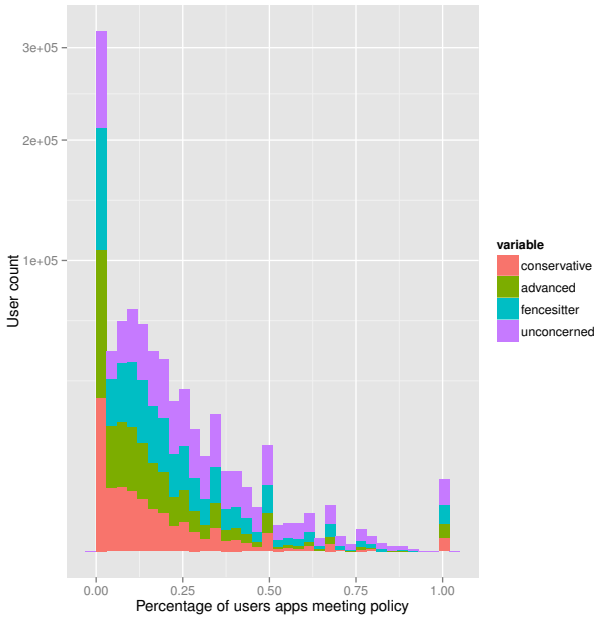

Figure 1: Adoption of policies modelling user behaviour.

4. DISCUSSION

In Figure 1 we can see that whilst the majority of users rarely follow these policies there are a few users who do seem to be installing apps on the basis of these policies most of the time.

[1]Users are replaced with incrementing numbers, apps with hashes.

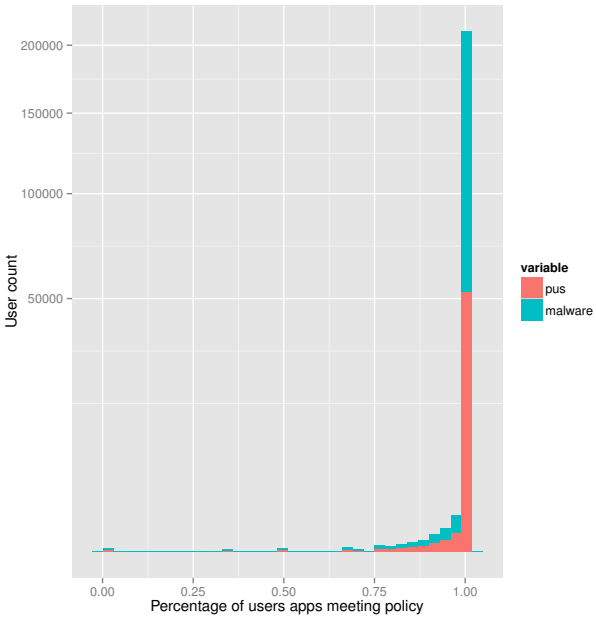

Figure 2: Adoption of policies for preventing PUS and malware.

In Figure 2 we see that most users do not install malware. If we consider PUS separate from malware however, the number of users without any PUS or malware is quartered. In the Carat data set we estimate that around 2% of users have any form of malware or PUS installed, and roughly 0.5% of users have malware installed.

Next steps will involve a trial creating an app store for Android with AppPAL built in. This will allow us to trial the system with users, and expert users to begin exploring policies to address their concerns.

5. REFERENCES

[1] D Barrera, J Clark, D McCarney, and P C van Oorschot. Understanding and improving app installation security mechanisms through empirical analysis of android. *Security and Privacy in Smartphones and Mobile Devices*, pages 81–92, October 2012.

[2] M Y Becker, C Fournet, and A D Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Computer Security Foundations*, 2006.

[3] P G Kelley, L F Cranor, and N Sadeh. Privacy as part of the app decision-making process. In *Special Interest Group on Computer-Human Interaction*, pages 3393–3402, New York, New York, USA, April 2013. ACM Request Permissions.

[4] J Lin, B Liu, N Sadeh, and J I Hong. Modeling Users' Mobile App Privacy Preferences. *Symposium On Usable Privacy and Security*, 2014.

[5] A J Oliner, A P Iyer, I Stoica, and E Lagerspetz. Carat: Collaborative energy diagnosis for mobile devices. In *Embedded Network Sensor Systems*, 2013.

[6] W Prata, A de Moraes, and M Quaresma. User's

demography and expectation regarding search, purchase and evaluation in mobile application store - Work: A Journal of Prevention, Assessment and Rehabilitation - Volume 41, Supplement 1/ 2012 - IOS Press. *Work: A Journal of Prevention*, 2012.

[7] X Wei, L Gomez, I Neamtiu, and M Faloutsos. Permission evolution in the Android ecosystem. In *Anual Computer Security Applications Conference*, pages 31–40, New York, New York, USA, December 2012. ACM Request Permissions.