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 pick the apps they want to run. App stores offer a wide range of software for users to choose. Users decide which apps to use based on their own privacy preferences as well as security rules. Sometimes this is based on what the store shows them first [8]. Sometimes it is based on information about the app. For example the ratings and permissions they request [7].

Users fall into patterns when picking apps [5]. By capturing these patterns explicitly as policies they can be enforced automatically. This reduces the burden on users to decide which apps they want. Security-savvy users (or professionals) may design policies themselves. These could be shared with others or used in organisation-wide curated app stores.

2. APPPAL

We use the AppPAL authorization logic [4], a version of SecPAL [6], to write policies with a precise semantics. AppPAL describes policies for an app to be installable. Statements are relative to specific principals, enabling delegation relationships to be expressed. Delegation is natural in the app store setting: it captures trust relationships among the users, the stores, the developers, and security vendors who vet apps.

Alice, a user, can state that an app is installable:

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

She can specify that an app is only installable if some constraint (checkable by inspecting the app or through static analysis) is true. She can also delegate parts of the decision to third parties. They must state specific facts about the app for Alice to accept them. For example, suppose Alice only wants apps without the `LOCATION` permission, and satisfying her *not-malware* requirement. She trusts McAfee to decide whether the *not-malware* policy is met:

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

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

Our AppPAL prototype is implemented as a Java library and runs on the Java and Dalvik virtual machines.

[*]With thanks to Igor Muttik at McAfee for sharing Android malware reports and N. Asokan at Aalto University for sharing their Carat data and discussions.

3. POLICIES

In a user study of 725 Android users, Lin et al. found four patterns that characterise user privacy preferences for apps [5]. The *Conservative* users are uncomfortable allowing an app access to any personal data for any reason. The *Unconcerned* users felt okay allowing access to most data for almost any reason. The *Advanced* users were comfortable allowing apps access to location data but not if it was for advertising reasons. The largest cluster, *Fencesitters* were broadly against collection of personal data for advertising purposes but opinions varied. We wrote AppPAL policies to describe each of these behaviours as increasing sets of permissions:

Policy	C	A	F	U
GET_ACCOUNTS	✗	✗	✗	✗
ACCESS_FINE_LOCATION	✗	✗	✗	
READ_CONTACT	✗	✗	✗	
READ_PHONE_STATE	✗	✗		
SEND_SMS	✗	✗		
ACCESS_COARSE_LOCATION	✗			

These simplify the patterns discovered. But AppPAL constraints can describe richer policies than permission sets.

Like other vendors, McAfee classify malware into several categories. The *malicious* and *trojan* categories describe traditional malware. Other categories classify PUPs (potentially unwanted programs) such as aggressive adware. Using AppPAL we can write policies to differentiate between different kinds of malware, characterising users who allow dangerous apps or those who install merely "unsavoury" apps.

```
"user" says "mcafee" can-say
  "malware" isKindOf(App).
"mcafee" says "trojan" can-act-as "malware".
"mcafee" says "pup" can-act-as "malware".
```

4. EXPERIMENTS

We aim to demonstrate AppPAL as a language for modelling app installation policies. We want to measure how far users enforce app installation policies informally. Users have opinions about apps [5]: but are they acting on them?

4.1 Methodology

We want to test how well policies capture user behavior. Installation data was taken from a partially anonymized[1]

[1]Users are replaced with incrementing numbers, app names are replaced with hashes to protect sensitive names.

database of installed apps captured by Carat [1]. By calculating the hash of known package names we can see who installed what. The initial database has over 90,000 apps and 55,000 users. On average each user installed around 90 apps each; 4,300 apps have known names. Disregarding system apps (such as `com.android.vending`) and very common apps (Facebook, Dropbox, Whatsapp, and Twitter) we reduced the set to an average of 20 known apps per user. To see some variations in app type, we considered only the 44,000 users who had more than 20 known apps. Using this data, and the apps themselves taken from the Google Play Store and Android Observatory [3], we checked which apps satisfied which policies.

4.2 Results

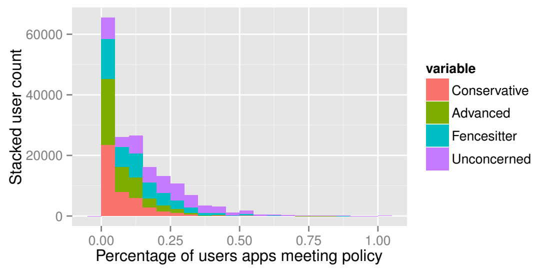

Figure 1: Use of policies modelling user behaviour.

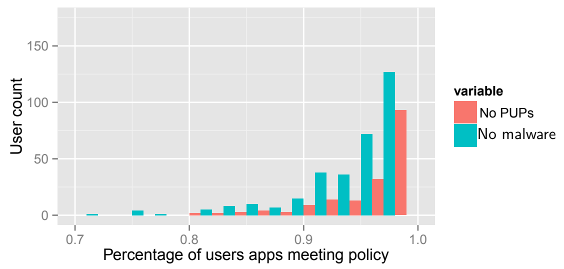

Figure 2: Percentage of malware in installed apps for users installing some malicious apps.

Figure 1 shows 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. For the unconcerned policy (the most permissive) only 1,606 users (4%) had 50% compliance; and only 120 users (0.3%) had 80% compliance. For the stricter conservative policy only 60 users were complying half the time, and just 7 more than 80% of the time. This suggests that while users may have privacy preferences the majority are not attempting to enforce them.

We found 1% of the users had a PUP or malicious app installed. Figure 2 shows that infection rates for PUPs and malware is small. A user is 3 times more likely to have a PUP installed than malware. Only 9 users had both a PUP and malware installed. Users who were complying more than half the time with the conservative or advanced policies complied with the malware or PUP policies fully. This suggests that users who pick their apps carefully are less likely to experience malware.

5. DISCUSSION

Most users seem to use apps irrespective of how uncomfortable they are with the permissions they request. A small set of users do seem to enforce these policies at least some of the time however. Exploring where this disconnect comes from is an avenue for future research. Do users not understand the relationship between apps and permissions [2]? Is enforcing them informally too difficult?

We claim that AppPAL can capture the differences in informal user policies. Using AppPAL we have written short policies describing user behaviour and used these to identify the users following them to varying degrees. Some limitations of our results include:

- We do not have the full user purchase history, and we can only find out about apps whose names match those in available databases. So a user may have apps installed that break the policy without us knowing.
- Recently downloaded apps used for experiment may not be the same version that users had, in particular, their permissions may differ. Permissions tend to increase in apps over time [9]; so a user may actually be more conservative than our analysis suggests.

To avoid these limitations, we want to test policies out directly with users, and to explore more complex policies dynamically. We will do this in two ways. First, by providing an Android app for AppPAL where a user can check their currently installed apps against predefined (perhaps downloadable) policies. Second, we want to enable automatically-curated app stores for Android that are parameterised on AppPAL policies.

6. REFERENCES

[1] A J Oliner et al. Carat: Collaborative energy diagnosis for mobile devices. In *Embedded Network Sensor Systems*, 2013.
[2] A P Felt et al. Android permissions: user attention, comprehension, and behavior. *Symposium On Usable Privacy and Security*, 2012.
[3] D Barrera et al. Understanding and improving app installation security mechanisms through empirical analysis of android. *Security and Privacy in Smartphones and Mobile Devices*, 2012.
[4] J Hallett et al. Towards an authorization framework for app security checking. In *ESSoS Doctoral Symposium*, 2014.
[5] J Lin et al. Modeling Users' Mobile App Privacy Preferences. *Symposium On Usable Privacy and Security*, 2014.
[6] M Y Becker et al. SecPAL: Design and semantics of a decentralized authorization language. *Computer Security Foundations*, 2006.
[7] P G Kelley et al. Privacy as part of the app decision-making process. In *Special Interest Group on Computer-Human Interaction*, 2013.
[8] W Prata et al. User's demography and expectation regarding search, purchase and evaluation in mobile application store. *Work: A Journal of Prevention*, 2012.
[9] X Wei et al. Permission evolution in the Android ecosystem. In *Anual Computer Security Applications Conference*, 2012.