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 preferences as well as security and privacy policies. Sometimes this is based on what the store shows them first [8]. Sometimes it is based on information about the app—the ratings and permissions they requests [7].

Users fall into patterns when picking apps [5]. By capturing these policies explicitly they can be enforced by a machine. This lessens the burden on users to decide which apps they want. 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. APPPAL

We have used the AppPAL authorization logic [4] to write policies. AppPAL (based on SecPAL [6]) can describe a policy for an app to be installable. AppPAL statements are made by principals. This enables delegation relationships to be expressed in policies. Delegation is natural in the app store setting. Trust relationships are made amongst the users, the stores, the developers, and security vendors vetting apps in the stores. AppPAL is implemented as a Java library and runs on the Java and Dalvik virtual machines.

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 a constraint (fact 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 Alice only wants apps without the `LOCATION` permission that meet her *not-malware* policy. 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 0
  "not-malware-policy" isMetBy(App).
```

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

3. POLICIES

Lin et. al. showed that users fall into four patterns when thinking about apps [5]. Conservative users are uncomfortable allowing an app access to any personal data for any reason. Unconcerned users felt okay allowing access to most data for almost any reason. Advanced users were comfortable allowing apps access to location data but not if it was for advertising reasons. Fence sitters (the largest cluster) were broadly against collection of personal data for advertising purposes but opinions varied. We wrote AppPAL policies to describe each of these behaviours as a simplification of [5]. The policies are of increasing strictness, each banning more 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. There are other categories for PUS (potentially unwanted software). Using AppPAL we can write policies to differentiate between different kinds of malware. This lets us explore the difference between users who install dangerous apps and those who happen to install unsavoury apps.

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

4. EXPERIMENTS

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

4.1 Methodology

We want to test how well these policies capture user behavior. Installation data was taken from a large, partially anonymized[1], database of installed apps [1]. By calculating the hash of a package name we can see who installed what.

From an initial database of over 90,000 apps, and 55,000 users. On average each user installed around 90 apps each, we unmasked 4,300 apps. Disregarding system apps (such as `com.android.vending`) and very common apps[2] we reduced the set to an average of twenty apps per user. We then removed users for whom we knew less than 20 apps; for whom we didn't have a representative sample of installations. Using this data, and the apps themselves taken from the Google Play Store and Android Observatory [3] we checked which apps satisfied which policies.

Our assumptions include:

- Downloaded apps used for experiments are not the same version as those used by users. Installed apps may have different permissions to the ones we looked at. Permissions tend to increase in apps over time [9]. 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 could only unmask a subset of installed apps. 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.

4.2 Results

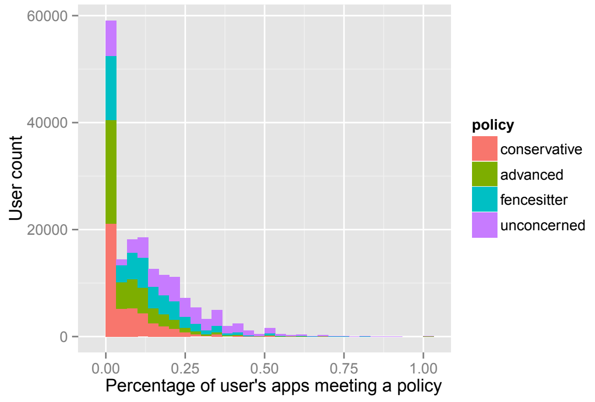

Figure 1: Use of policies modelling user behaviour.

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. This suggests that while users may have privacy preferences the majority are not attempting to enforce them.

We found 2% of the users had a PUS or malicious app installed. In Figure 2 we see that of those users with a malicious app installed PUS accounts for around half of the malware infections.

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 too difficult?

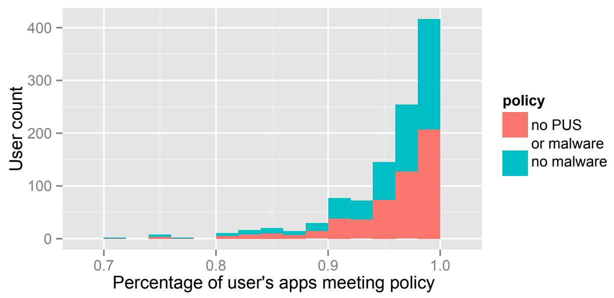

Figure 2: Percentage of malware in installed apps for users installing some malicious apps

AppPAL can capture the differences in user policies. AppPAL can filter apps on the basis of these policies. Using AppPAL we have written short policies describing user behaviour and been able to identify users following them to varying degrees.

Next steps will involve a trial creating an app store for Android with AppPAL built in. This will allow us to trial policies with users; and begin testing more complex policies dynamically. To begin testing AppPAL we plan an Android app where a user can check their currently installed apps against several 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.* University of Edinburgh, February 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.

[1]Users are replaced with incrementing numbers, app names with hashes.
[2]Specifically Facebook, Dropbox, Whatsapp, and Twitter.