Ch 2
Draft 1

Ok but needs some improvement
— SecPAL
— Stores
— delegation, roles, principles

Authorization Logic For Mobile Ecosystems

Joseph Hallett

Doctor of Philosophy

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

2017

Abstract

TODO

Table of Contents

List of Figures

List of Tables

Didn't you have a picture at one point? Useful to show users, stores, devices, developers

Chapter 2

Background

principles in AppPAL core to thesis (maybe it's in another chap but IMO belongs early)

2.1 The mobile ecosystem

seems a bit silly

This dissertation talks about the policies surrounding *mobile ecosystems*; but what, precisely, do we mean by the *mobile ecosystem*? A cow after all is both mobile and has an ecosystem of grass, farmers, and other cows surrounding it. Succinctly when we refer to the mobile ecosystem we are referring to **the interactions surrounding the use of smart phones and tablet computers**. To fully understand the scale of the ecosystem however, it is best to describe some of the history surrounding it.

2.1.1 Mobile devices: the story so far

Some of the earliest mobile devices were the Personal Data Assistants (PDAs) devices of the early 90s. The earliest devices, such as Apple's Newton, were portable miniature computers designed to store personal information, calendars and notes. Developers could even program additional apps for them (in the case of the Newton using Apple's Dylan language). At the time mobile phones were just portable telephones, but starting with the IBM Simon in 1994 these devices started to have some of the functionality of the PDA, becoming what would later be called a *smart phone*. The big advantage of these early smart phones over the PDA was they had a telephone connection. The earliest devices allowed their users to access email and fax, as well as managing personal information. The early smart phones were somewhat underpowered compared to the PDA devices so both continued to develop alongside one

another, and started to become more affordable.

In 1998 the Symbian OS was released. Developed by Psion (who made PDAs), and Motorola, Ericsson and Nokia (all phone manufacturers). By the early 2000s it would start to become the dominant smart phone OS. These devices could install apps, written in C++ or Java (if the phone supported JME). They had cameras, could play music. They even had early malware which would illicitly send texts to premium rate numbers. They were the forebears of the *modern* smart phone. They were also starting to become affordable, with the lowest end models starting to being affordable by children and teenagers[1]. Devices like Nokia's N-Gage were marketed directly towards these younger users and featured games users could buy for their devices.

In the mid-2000s we first start to see the mobile ecosystem proper. We have users with different devices, downloading apps from different sources (not all legitimate). These devices contained personal information of the PDAs, but also photographs and music. Users could browse the web, and send each other pictures. With increasing amounts of personal information malware authors started to take note. In a blog post from 2006 for Symantec, Chien noted [8]:

> "While threats exist and are actively spreading, we are probably still years away from the situation we have with the Microsoft Windows [···] We have already seen spyware applications for mobile devices (e.g. Spyware.Flexispy) that can monitor activities on the mobile device and then send them to a remote server. [···] Just as worrying is the fact that the adware market is just beginning to take notice of mobile devices. Already some Bluetooth advertising schemes have been tested, where a bus stop is outfitted with a device that just spams out messages via Bluetooth.
>
> [···]
>
> So, while worms and Trojans already exist for the mobile platforms, spyware and adware applications are just now gaining a foothold in the mobile device space. Spyware and adware pose a potentially large security issue in the near future, as the companies that produce such applications are less affected by the natural limiting factors."

In 2008 Apple released the first iPhone, and shortly after Google released the Android OS. Symbian would release its final version in 2012, with Android as the dominant OS on most devices. These operating systems differed from past efforts, and conventional desktop OS, in that they were far more controlled

[1] If you saved up for what seemed like forever...

than previous systems. Apple's devices cannot run un-vetted code or install software from outside of its App Store, which Apple controlled. Android devices offered a similar, though less heavily vetted, app marketplace [21]. Whilst Android users could install software from other sources, the option was hidden by default. As well as restrictions to software, these devices were also better protected. Devices no longer had an all powerful *root* account. APIs were provided that restricted malicious behaviors. One example was stopping SMSs being sent programatically to premium rate accounts; essentially ending a monetization technique that had been prevelent in malware before.

With iOS and Android users became more aware about privacy on their devices. A survey in 2012 by Chin et al. found that users were more concerned about privacy on their mobile devices, than on their laptops [5]. Users of a smartphone are more likely to install an app that came recommended from a friend, was popular or was free. The drive for ever cheaper apps, particularly on Android where users were even less likely to pay for software, lead to many developers adding adware and tracking libraries into their apps. These libraries accomplished a variety of tasks ranging from crash and error tracking, to collecting personal data to be sold to advertisers to subsidize the app's price [24].

This dynamic between users who were increasingly concerned about their privacy and apps which were increasingly privacy invading, lead many to argue that the mobile device operating systems needed better controls for what data and permissions an app could have [16]. Many different schemes were suggested to fix this and provide finer privacy controls [13, 3, 6, 1] (some of which will be described further in **??**). In general, however, users did not really understand the how app permissions worked [10]. By 2017 (and the present day) iOS and Android had settled on a model where user's were asked by their devices if an app could access certain data when it first requested it and then the user could revoke that decision later if they wished.

Mobile devices became more ubiquitous, and PDAs reappeared now called *tablets* or *iPads*. These devices were identical to the smart phones, but larger and they generally did not have a cellular data connection instead using wifi. They couldn't send texts or make phone calls but a shift away from traditional SMS and cellular phone calls to internet backed communications such as WhatsApp, iMessage, Skype and FaceTime meant these devices were essentially

interchangeable with smart phones.

Advances in secure co-processors meant that mobile devices are even more capable. Banks allow users to link their devices to their bank accounts and use their device, or even the new *smart watches* as a debit card. This quickly became a common and ubiquitous payment method quickly with even street vendors, who hadn't taken cards previously, accepting contactless payment via a mobile device.

Mobile devices record increasingly personal information. Smart watches record and share a user's pulse with others. Mobile health apps are made to track and monitor medical conditions. In the US, the Health Insurance Portability and Accountability Act (HIPAA) requires that healthcare providers transmit medical information securely, but many apps have basic security problems [9], and many of the healthcare apps do not handle the medical information securely [14].

2.1.2 The Mobile Ecosystem and the need for Policies

With mobile devices becoming increasingly capable and managing ever-increasing amounts of information there is a need to manage how the devices behave. As users become aware of the privacy implications of their devices, they express preferences about which apps they want to install [23], even if they do not follow through on their preferences [12]. Employees now bring their mobile devices to work and use them to access company email and documents. In response to this companies publish mobile device policies that describe how the devices should be used within the company. They may also use mobile device management (MDM) software to enforce the policies. If a company wishes to use an app for business purposes there may be regulation they need to follow, such as HIPAA.

These policies vary in terms of formality. A user may never write their privacy preferences in a formal language, but they may make decisions guided by them. For example which apps to install and which to avoid. They may make decisions based on what their friends have told them, or what a review said about the app. By describing the policy in a formal language we can start to express the policy more rigorously. We can start to make comparisons between users, and with rules for checking the policy start to help the user

to make decisions more accurately, or measure the extent a user follows their stated policy [12].

A company looking to control the mobile devices their employees bring to work might write a bring your own device (BYOD) policy. They might also use MDM software to control some aspects of their devices. The company might write these with varying degrees of formality but often they are written using natural language. This adds vagueness and can lead to confusion as to how a policy should be implemented. Using formal languages we can model the policies precisely, helping clarify their meanings and make precise comparisons between different policies. We can tie the rules in the BYOD policies to the MDM tools used to implement them.

If a company needs to use apps which conform with a policy such as HIPAA they could use static analysis tools to check for some aspects of the policy. Perhaps the company might use Mallodroid [9] to detect when data is sent unencrypted. It is important, however, not to confuse the tools and techniques we might use to implement parts of a policy with the end goal of ensuring that the policy is followed. A formal language that lets us tie the policy to its implementation can help show how the policy is checked. It lets us see what rules from the policy are checked for by which tools, and identify gaps where the policy is not being checked.

When describing how mobile phones had evolved, we described how we arrived with two main operating systems for mobile devices: iOS and Android. The trust models in the systems differ, as do the assumptions about what an app can do when running on each system. There are also similarities. iOS and Android (from Android M) have a similar permissions system for apps. When people make comparisons between them, however, the differences are often vague. iOS is a *walled garden*. Android is *more open*. Again, by using formal languages we can start to make comparisons that are precise and show the differences clearly.

We need an authorization logic for mobile ecosystems.

2.2 SecPAL

SecPAL is an authorization language developed by Becker et al. to describe policies and delegation chains surrounding distributed services [2]. It was

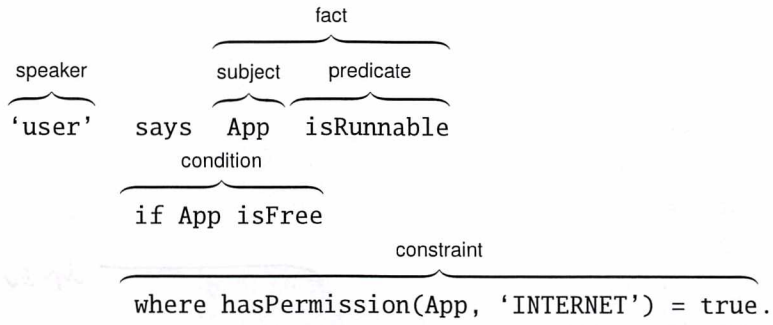

Figure 2.1: Structure of a SecPAL assertion.

designed as a high-level human-readable language that allowed the policy specification and maintenance to be separated from the implementation mechanisms.

SecPAL was designed to improve over previous high-level languages in several areas. It was designed to be more expressive than XrML [15], SPKI/SDSI [7], and Delegation Logic [17]. More readable than XACML [20] and other XML-based policy languages. Furthermore it was designed to be intuitive and unambiguous with its semantics unlike other languages (most notably XACML) which had natural language descriptions with ambiguous and inconsistent specifications that had been retrofitted to the language instead of being designed with the language in the first place [4, 22, 18].

Its original application was to model and enforce access control policies in grid computing systems [2]. In this dissertation we describe how the language can be extended to describe the policies surrounding the mobile ecosystem.

At its core, SecPAL is a language with a simple grammar (Figure 2.2) and three evaluation rules (Figure 2.3). The language's simplicity makes it easy to apply to a new domain by instantiating it with predicates and constraints that describe the domain. This simplicity does not come at the cost of its expressiveness. SecPAL supports delegation (by using *can-say* verbs), role and attribute based policies (by using *can-act-as* verbs) and arbitrary constraints.

SecPAL's semantics are given in Figure 2.4. A query (q) to a SecPAL program (a collection of facts and relationships called the *assertion context* or *AC*) asks if there exists a renaming (θ) such that the rules of SecPAL can be used to derive the, possibly renamed, query $(AC, \infty \vdash q\theta)$. The renaming must be

e	::=	x	*(variables)*
	|	A	*(constants)*
pred	::=	has | can | ...	*(predicates)*
D	::=	0	*(no delegation)*
	|	∞	*(delegation)*
vp	::=	pred $e_1 \ldots e_n$	*(verb phrase)*
	|	can-say$_D$ fact	
	|	can-act-as e	
f	::=	e vp	*(fact)*
claim	::=	f if $f_1, \ldots, f_n$; c	
assert	::=	e says claim.	
AC	::=	$assert_1 \ldots assert_n$	*(assertion context)*
c	::=	$\top$	*(no constraint)*
	|	$e'_1 = e'_2$	*(constraints)*
	|	...	
e′	::=	e | function($e_1, \ldots e_n$)	

Figure 2.2: BNF description of SecPAL.

$$\frac{(A \text{ says } f \text{ if } f_1 \cdots f_n \text{ where } c) \in \text{AC} \quad \forall i \in [1 \cdots n].\text{AC}, D \models A \text{ says } f_i\theta \quad \vdash c\theta \quad vars(f\theta) = \emptyset}{\text{AC}, D \models A \text{ says } f\theta} \text{ cond}$$

$$\frac{\text{AC}, \infty \models A \text{ says } B \text{ can-say}_D\ f \quad \text{AC}, D \models B \text{ says } f}{\text{AC}, \infty \models A \text{ says } f} \text{ can-say}$$

$$\frac{\text{AC}, D \models A \text{ says } x \text{ can-act-as } y \quad \text{AC}, D \models B \text{ says } y\ vp}{\text{AC}, D \models A \text{ says } x\ vp} \text{ can-act-as}$$

Figure 2.3: SecPAL's evaluation rules.

$AC, \theta \vdash q$	Defining relation. A query assertion q is valid given the assertions contained in the assertion context AC and a variable substitution θ.
ϵ	The empty substitution.
1. $AC, \theta \vdash e$ says $fact$	if $AC, \infty \models e\theta$ says $fact\theta$ and $dom(\theta) \subseteq vars(e$ says $fact)$
2. $AC, \theta_1\theta_2 \vdash q_1, q_2$	if $AC, \theta_1 \vdash q_1$ and $AC, \theta_2 \vdash_2 q_2\theta_1$
3. $AC, \theta \vdash q_1$ or q_2	if $AC, \theta \vdash q_1$ or $AC, \theta \vdash q_2$
4. $AC, \epsilon \vdash$ not(q)	if $AC, \epsilon \nvdash q$ and $vars(q) = \emptyset$
5. $AC, \epsilon \vdash c$	if $\models c$

Figure 2.4: SecPAL's semantics as described by Becker [2].

specific and talk only of the variables contained in the query (third statement in Figure 2.4), conjunction and disjunction work as would be expected (fourth and fifth statements). Negation is only allowed in an extremely limited form (sixth statement): a query can be not true if it contains no variables and there is no way to show that the AC supports the query. This limited form means that queries remain decidable: the rules inside the AC are not permitted to use negation.

SecPAL was later extended to add universal quantification, and the possibility of dynamic assertion retrieval (they define a safety condition, but don't describe a protocol for retrieving assertions) [19]. These ideas would be expanded to create a new language called DKAL [11]. Gruevich et al. showed how SecPAL policies could be translated into DKAL policies [11] so any SecPAL-based policies could be updated to DKAL. We don't use these additional features in this thesis as we did not need the additional expressiveness they provide to adequately describe the policies in the mobile ecosystem.

Bibliography

[1] Michael Backes, Sebastian Gerling, Christian Hammer, Matteo Maffei, and Philipp von Styp-Rekowsky. AppGuard Enforcing User Requirements on Android Apps. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 543–548. Springer, Berlin, Heidelberg, March 2013.

[2] Moritz Y. Becker, Cdric Fournet, and Andrew D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, January 2010.

[3] Alastair R. Beresford, Andrew Rice, Nicholas Skehin, and Ripduman Sohan. MockDroid: Trading Privacy for Application Functionality on Smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, pages 49–54, New York, NY, USA, 2011. ACM.

[4] Jery Bryans. Reasoning About XACML Policies Using CSP. In *Proceedings of the 2005 Workshop on Secure Web Services*, SWS '05, pages 28–35, New York, NY, USA, 2005. ACM.

[5] Erika Chin, Adrienne Porter Felt, Vyas Sekar, and David Wagner. Measuring User Confidence in Smartphone Security and Privacy. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 1:1–1:16, New York, NY, USA, 2012. ACM.

[6] Mauro Conti, Vu Thien Nga Nguyen, and Bruno Crispo. CRePE: Context-Related Policy Enforcement for Android. In Mike Burmester, Gene Tsudik, Spyros Magliveras, and Ivana Ili, editors, *Information Security*, number 6531 in Lecture Notes in Computer Science, pages 331–345. Springer Berlin Heidelberg, October 2010. DOI: 10.1007/978-3-642-18178-8_29.

[7] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. Technical Report RFC 2693, IETF, 1999.

[8] Eric Chien. Spyware and Adware on Mobile Devices, May 2006.

[9] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgrtner, Bernd Freisleben, and Matthew Smith. Why Eve and Mallory Love Android: An Analysis of Android SSL (in)Security. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 50–61, New York, NY, USA, 2012. ACM.

[10] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android Permissions: User Attention, Comprehension, and Behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 3:1–3:14, New York, NY, USA, 2012. ACM.

[11] Y. Gurevich and I. Neeman. DKAL: Distributed-Knowledge Authorization Language. In *2008 21st IEEE Computer Security Foundations Symposium*, pages 149–162, June 2008.

[12] Joseph Hallett and David Aspinall. AppPAL for Android. In *Engineering Secure Software and Systems*. Springer Verlag, April 2016.

[13] Jinseong Jeon, Kristopher K. Micinski, Jeffrey A. Vaughan, Ari Fogel, Nikhilesh Reddy, Jeffrey S. Foster, and Todd Millstein. Dr. Android and Mr. Hide: fine-grained permissions in android applications. In *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, 2012.

[14] Konstantin Knorr, David Aspinall, and Maria Wolters. On the Privacy, Security and Safety of Blood Pressure and Diabetes Apps. In *ICT Systems Security and Privacy Protection*, pages 571–584. Springer, Cham, May 2015.

[15] Vladimir Kolovski. *Logic-Based Access Control Policy Specification and Management*. Department of Computer Science, University of Maryland, 2007.

[16] Ilias Leontiadis, Christos Efstratiou, Marco Picone, and Cecilia Mascolo. Don't Kill My Ads!: Balancing Privacy in an Ad-supported Mobile Application Market. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, HotMobile '12, pages 2:1–2:6, New York, NY, USA, 2012. ACM.

[17] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation Logic: A Logic-based Approach to Distributed Authorization. *ACM Trans. Inf. Syst. Secur.*, 6(1):128–171, February 2003.

[18] Massimiliano Masi, Rosario Pugliese, and Francesco Tiezzi. Formalisation and Implementation of the XACML Access Control Mechanism. In Gilles Barthe, Benjamin Livshits, and Riccardo Scandariato, editors, *Engineering Secure Software and Systems*, number 7159 in Lecture Notes in Computer Science, pages 60–74. Springer Berlin Heidelberg, February 2012. 00029 DOI: 10.1007/978-3-642-28166-2_7.

[19] Moritz Y Becker. SecPAL: Formalization and Extensions. Technical Report MSR-TR-2009-127, Microsoft Research, September 2009.

[20] OASIS. eXtensible Access Control Markup Language (XACML) Version 3.0. Technical report, OASIS, January 2013.

[21] Jon Oberheide and Charlie Miller. Dissecting the Android Bouncer, 2012.

Is this most recent now?

[22] Carroline Dewi Puspa Kencana Ramli, Hanne Riis Nielson, and Flemming Nielson. The logic of XACML. *Science of Computer Programming*, 83:80–105, April 2014.

[23] Norman Sadeh, Jason Hong, Lorrie Cranor, Ian Fette, Patrick Kelley, Madhu Prabaker, and Jinghai Rao. Understanding and Capturing People's Privacy Policies in a Mobile Social Networking Application. *Personal Ubiquitous Comput.*, 13(6):401–412, August 2009. 00202.

[24] Seungyeop Han, Jaeyeon Jung, and David Wetherall. A Study of Third-Party Tracking by Mobile Apps in the Wild. Technical Report UW-CSE-12-03-01, University of Washington, 2012.