# Authorization Logic for Mobile Ecosystems

**Third Year Report**

Joseph Hallett

September 26, 2016

This report describes the progress made in the third year of my PhD. It describes my work this year on BYOD policies and encoding them in AppPAL, a policy language for the Android mobile ecosystem, as well as work developing AppPAL policy analysis tools and a probabilistic variant of AppPAL, as well as plans for how this should continue in the time remaining. The report goes on to describe my plans for the thesis and gives a table of content as well as a plan for submission by June 2017.

## 1. Introduction

Users can choose from a large number of apps for their phones. Their devices have access to a huge amount of personal information, and despite improvements giving users more control over what information they share [1, 2], users still have to be careful to ensure their privacy and data-security preferences are followed. These devices are increasingly entering every part of their user's lives; being used for work as well as for home.

The interactions between people and mobile devices requires an increasing number of trust relationships and policies—rules that describe how a device or user should behave. These trust relationships take the form of delegations of responsibilities to third-parties. For example a user might delegate to an store to check that their apps are up to date and not malicious.

These policies and trust relationships are often ad-hoc and informally specified. Users are not clear about how they want their apps to behave [21]. Businesses specify policies for users' devices in their workplace using natural language. Different app marketplaces have different policies describing how their users and developers can use their store and what rights they have, but these are hidden behind long EULAs and developer agreements. The interactions between users, their devices, their employers, the stores they buy apps from and the apps themselves forms a vibrant mobile ecosystem of requirements, preferences and trust relationships for users who are often left to understand and enforce their policies on their own.

Authorization logics are designed to express policies precisely and make decisions on the basis of facts and have been used to implement access control systems [23] (deciding whether someone has access to a file or not). By using these formal systems we can make informal policies in the mobile ecosystem more precise, we can start to automate enforcement and make precise comparisons between the different entities in the ecosystem and their roles and responsibilities.

## 1.1. Goals of this PhD

The goal of this work is to create a practical scheme for linking the high-level human policies in the mobile ecosystems, to the technical means for enforcing policies. The work will focus on the ecosystems surround Google's Android operating system, the largest mobile OS with over 80% market share [17]. Unlike iOS (the OS for Apple's devices), there is a greater variation in the app stores available (Play Store, Amazon App Store, Yandex, Baidu and 360 Mobile Assistant to name but a few) as well as a broader amount of research on Android's apps and permission systems [11, 8, 21].

Much prior work has focused on the technical side of policies; extending Android with more sophisticated permission systems and complex enforcement schemes [18, 5, 16]. This work differs in that it focuses on the human policies, and is happy to delegate the precise enforcement mechanism to a tool.

To do this I have taken an existing authorization language, SecPAL [7], and applied it to the problem of formalising the user level policies in mobile ecosystems (and in doing created a modified implementation of SecPAL called *AppPAL* [13]) Specifically I aim to show:

> **Thesis:** *Formal languages can model the natural language policies in the Android ecosystem, and in doing so help users, businesses, and developers make decisions on the basis of a declarative policy, rather than an informal understanding of their preferences, rules, and policies.*

- That authorization languages with named speakers (specifically AppPAL) are a good fit for describing the policies and trust relationships surrounding Android.
- That users have policies but that they struggle with enforcing them in practice.
- That AppPAL can be used to build systems where policies can be enforced by delegation to tools, and the details of implementing such a system.
- To show how AppPAL can be used to describe precisely the differences between different policies (be they BYOD policies in companies, developer agreements in stores or app preferences), and identify problems with them ahead of time.

## 2. Progress

Last year, I worked on implementing AppPAL on Android, built a security knowledge base to store results from static analysis checkers and metadata about apps. I surveyed user and developer policies present in stores and their distribution mechanisms. I also

$$\begin{array}{rcl} \langle \text{E} \rangle & := & \langle \text{Variable} \rangle \mid \textbf{'constant'} \\ \langle \text{Variable} \rangle & := & \textbf{(Type:)}?\textbf{VariableName} \end{array}$$

Figure 1: Changes to SecPAL's variable syntax. Additions are shown in red.

investigated whether users install apps matching their privacy preferences and presented a draft of a paper, that was later accepted and published [13].

This year I proposed working on a case-study looking at BYOD policies and developing a knowledge distribution protocol to define how AppPAL assertions should be distributed. Work on the BYOD policies has resulted in a presentation at iFM's doctoral symposium [14] (the short paper is included in Appendix C), but there hasn't been progress working on a knowledge distribution protocol. Instead, I have been worked on developing analysis tools for AppPAL policies and started creating a probabilistic variant of AppPAL.

## 2.1. Typed AppPAL

SecPAL's safety condition requires all variables in an assertion's head to be used in the body. When trying to describe policies a common pattern is to say a variable in an assertion belongs to a certain domain. For example, Alice might be willing to let her friends say what apps are suitable for her children. This could be expressed in SecPAL as follows:

```
'alice' says Friend can-say App isSuitableFor(Child)
  if Friend isFriend, App isApp, Child isChild.
```

The conditionals in this assertion add unnecessary noise to the assertion. We know from the names of the variable what set of constants might be used to instantiate it (Alice's *friends*, *apps* or *children*). To avoid this noise I added a sugared syntax to AppPAL that allows variables to declare their *type*. Using the sugared notation the above statement becomes:

```
'alice' says Friend:F can-say App:A isSuitableFor(Child:C).
```

The changes to SecPAL's syntax is shown is Figure 1. After an assertion has been parsed the variables with types are extracted for each variable a conditional is added that `VariableName` *is* `Type`, and the types are removed from the variables.

This gives a cleaner policy language however it also means that the predicates used start to have some intrinsic meaning. A predicate starting with `is` describes some property of the predicate's subject. SecPAL did not require predicates follow any naming conventions, however with AppPAL we have started to give predicates meaning based on their name. This is expanded in the subsection 2.2 where we describe other predicate prefixes that we use for specific purposes.

Listing 1: Procedure used to expand types from AppPAL into SecPAL.

```
def expand_types(a: Assertion) -> Assertion:
  for v in a.head.vars():
    if v.type != None:
      f = Fact(v, 'is'++str(v.type))
      a.body.add(f)
  return a
```

## 2.2. BYOD Policies

Employees increasingly bring their own devices to work. Around 70% of all companies having some form of bring your own device (BYOD) scheme [29]. In hospitals the use of personal devices as part of patient care is even more common. One survey from the UK found that around 80% of surgical doctors were willing to use their personal devices for work, with 85% regularly using them to look up information, and 50% having medical apps already installed on their phones [26]. Another survey from the US found similar numbers of physicians regularly using their devices as part of their work [25].

To implement policies companies can use mobile device management (MDM) software to try and implement their rules. MDM solutions are offered by a many companies, including IBM (MaaS360), VMware (AirWatch) and MobileIron; however they all offer a comparable set of features, namely:

1. App wrapping; where an app is rewritten to offer some addtional security or network properties. This is often limited to routing all the app's network traffic through a VPN.

2. Basic security configuration; where security settings can be turned on and off to require encryption (for example) on a device.

3. Provisioning; where IT departments can install and update apps and their configuration files. Email and LDAP configuration is common.

4. A curated app store; where IT departments can white or black list apps.

Whilst MDM software can enforce some policies; having an MDM package does not guarantee a company's BYOD policy is enforced correctly, or even enforced at all. One survey of companies using MDM software found that over half of them had devices that broke their policies within their networks [24]. Whilst the majority of BYOD policies are relatively simple: a report from the healthcare sector, where policies are typically more complex, noted that:

> "an approach toward educating the users instead of trying to control the technology may be more practical..." [25]

The tension between MDM software and having something that employees can understand has possibly led to many BYOD policies being published using natural language. Whilst there has been much research on the *technical* means to enforce BYOD policies [22, 6, 9] there has been little work linking the natural languages of the policies to the implementation used to enforce the rules, as well as making precise comparisons between policies.

I have taken policies from five different sources as part of a case study into the rules and idioms used in BYOD policies. One is from the SANS institute and serves as a guide for companies seeking to implement their own BYOD policy, one is from HiMMS and is a guide for hospitals looking to implement a BYOD policy. Another from an NHS trust is a complex policy describing how medical staff should use their own devices when dealing with patients, as well as BYOD concerns. The other two are simple BYOD polices typical of what most companies implement and are taken from Edinburgh University and a company selling emergency sirens.

I have used AppPAL to create a formalisation of each of the policies. This has helped identify common predicates used in each of the policies (Table 1). It has also led to a set of standard prefixes for AppPAL predicates. In subsection 2.1, I described how predicates that start with *is* are reserved for predicates that give a type to their subject. Through translating the BYOD policies I found four distinct categories of predicate that could encode all the different rules in each of the policies as well as many similarities in concerns for BYOD policies.

## Future Directions

I have gone as far as translating the policies, and ensuring they use semi-standard terms and AppPAL predicates, and have also developed some rudimental tools for visualising the policies and what responsibilities each speaker has. I plan to make comparisons between each of the policies and describe a standard set of predicates and tooling for reasoning about BYOD policies.

This work should ideally lead to a publication, perhaps at iFM[1] where earlier work on this was presented in their doctoral symposium [14].

All the BYOD policies I have looked at make use of acknowledgments by the subjects of the policy that they will follow, or be aware of, certain rules. These are interesting as they cannot be enforced by tooling (which looks at technical means to prevent devices from performing actions). These acknowledgments may carry certain legal responsibilities; it would be interesting to look at how these can be collected, what delegations of trust are made within them, and what responsibilities each speaker has. Working out what responsibilities a speaker has and what delegations are made is, of course, recoverable from the trace of a dynamically evaluated AppPAL policy. It would be nice however to do this statically.

I had hoped to develop some analysis tools that might spot problems with the policies automatically (described in subsection 2.3) but the current state of BYOD policies is

[1]Based on last year's CFP the submission date would be early January.

| Predicate | Code3PSE | Edinburgh | HiMMS | NHS | SANS | Description |
|---|---|---|---|---|---|---|
| Device `canBackupTo`(Server) |  | ✓ | ✓ |  |  | Says the device may send backups to a server. |
| Device `canCall`(Number) |  |  |  | ✓ | ✓ | Says the telephone numbers a device can call. |
| Device `canConnectToAP`(AP) |  |  | ✓ |  | ✓ | Says a device may associate with an access point. |
| Device `canConnectToNetwork`(Subnet) | ✓ |  |  |  | ✓ | Says the device may connect to a network, for example computers all within a subnet. |
| Device `canConnectToServer`(IP) |  |  |  | ✓ | ✓ | Says the subject (a device) can connect to a given server (identitfied by a URL). |
| Device `canInstall`(App) |  |  |  | ✓ | ✓ | Says the device may install an app. |
| User `canMonitor`(Device) | ✓ |  |  | ✓ | ✓ | Says the subject (a user) can monitor and unlock a device. |
| Device `canStore`(File) |  |  | ✓ |  | ✓ | Says a device may store some data in permanent (non-transient) storage. |
| User `canUse`(Device) |  |  |  | ✓ | ✓ | Says a user may use a device. |
| User `hasAcknowledged`(Policy) | ✓ | ✓ | ✓ | ✓ | ✓ | Says an individual has acknoweldged a policy. |
| Device `hasFeature`(Feature) |  | ✓ |  | ✓ |  | Says a device has a feature enabled. |
| Something `hasMet`(Policy) |  | ✓ |  | ✓ | ✓ | Says a something has met a given set of requirements. |
| Device `isActivated` |  |  | ✓ | ✓ | ✓ | Specifies a device has been activated for BYOD usage. |
| App `isApp` | ✓ |  | ✓ | ✓ | ✓ | Specifies an app. |
| Something `isApproved` |  |  | ✓ |  | ✓ | Specifies that something has been vetted and approved of. |
| Data `isData` | ✓ |  |  |  | ✓ | Specifies data. |
| Device `isDevice` | ✓ | ✓ | ✓ | ✓ | ✓ | Specifies a device. |
| User `isEmployee` | ✓ | ✓ |  | ✓ | ✓ | Specifies that someone is an employee. |
| Device `isEncrypted` |  | ✓ |  | ✓ | ✓ | Specifies a device is encrypted. |
| Feature `isFeature` | ✓ |  |  |  | ✓ | Specifies a feature. |
| App `isInstallable` |  |  |  | ✓ | ✓ | Specifies an app is installable. |
| Device `isLost` | ✓ |  | ✓ | ✓ | ✓ | Specifies a device is missing. |
| Device `isOwnedBy`(User) | ✓ | ✓ | ✓ | ✓ | ✓ | Specifies something's owner. |
| File `isSecurityLevel`(Level) |  |  | ✓ |  | ✓ | Specifies some data as having business sensitive information. |
| Something `isString` | ✓ |  |  | ✓ |  | Specifies a string. |
| Number `isTelephoneNumber` |  |  |  | ✓ | ✓ | Specifies a telephone number. |
| Device `mustDisable`(Feature) | ✓ | ✓ |  | ✓ | ✓ | Forces the disablement of a feature (or a device). |
| User `mustInform`(User, Something) | ✓ |  | ✓ | ✓ | ✓ | Forces someone to disclose something to someone. |
| Device `mustWipe` | ✓ |  | ✓ | ✓ |  | Forces a device to be erased. |

Table 1: Summary of predicates used in multiple BYOD policies.

| Prefix | Meaning |
|---|---|
| subject `canAction` | The subject is allowed to perform the action. |
| subject `hasAction` | The subject has performed the action. |
| subject `isProperty` | The property holds true for the subject. Mostly used for typing statments. |
| subject `mustObligation` | The subject is required to satisfy the obligation |

Table 2: Standard prefixes for AppPAL predicates

that most are too simple to have problems, and the faults in longer policies are limited to rules being duplicated or underspecified rather than more interesting problems, such as having multiple ways of satisfying a rule which can undermine the security goals.

## 2.3. Automatic Analysis of Policies

When examining an AppPAL policy it is natural to wonder whether the policy is as optimal (in terms of the rules and facts required to decide a query, with respect to the number of rules in the policy) as it could be, or whether a decision is even reachable given the rules and facts contained in the policy. Does an assertion context (i.e. the assumptions available to prove a statement) contain enough statements to use a given rule? If there are multiple ways of deciding whether some statement is true or not does one rule require fewer statements than any other? Does one rule require only a subset of the facts of another rule, implying the second is redundant?

### 2.3.1. Checking Satisfiability

Inference in AppPAL happens by collecting ground facts and constraints that satisfy rules. These rules are combined to form a policy. When a rule is satisfiable there is a combination of facts that could satisfy its conditionals. If there are no facts that could satisfy the rule then the policy may be incomplete as there are rules that can never be used. Formally for any goal $G$:

$$\begin{aligned} G \in \text{Satisfiable } \textit{if } \exists A \in \text{AC} :\ \exists \theta :\ & G \equiv \text{head}(A\theta) \\ & \wedge (\text{conditionals}(A) = \emptyset \\ & \quad \vee\ \forall G' \in \text{conditionals}(A).\ G' \in \text{Satisfiable.}) \end{aligned}$$

This is a similar idea to the notions of *satisfiability* in Datalog (and more generally logic programming). Satisfiability in Datalog is defined as [3]:

> **Satisfiability:** An IDB predicate $s$ of a program $P$ is *satisfiable* if there is some EDB $D$, such that $P$ defines a non-empty relationship for $s$.

SecPAL does not necessarily[2] split it's assertion context into an EDB (extinsional database, essentially the ground facts) and IDB (intensional database, relations defined by rules). If a Datalog predicate is taken as a SecPAL goal, a Datalog program is taken as a number of SecPAL assertions, and the EDB is taken as AppPAL ground facts (i.e. assertions with no conditionals) then the definition is equivalent.

For an example of satisfiability, consider the following snippet taken from the NHS policy described in subsection 2.2. The rule described in the policy is that an app must be approved by the *integrated governance commitee* as well as by either the *care and clinical policies group* as well as the *management of information group* depending on whether it is for clinical or business use. We can describe this in AppPAL as follows:

[2]It is implementation dependant. If a translation to Datalog$^C$ was used as suggested by Becker [7] then it might.

```
'nhs-trust' says App isInstallable
  if App isApproved, App isUsableClinically.
'nhs-trust' says App isInstallable
  if App isApproved, App isUsableNonClinically.
'nhs-trust' says 'igc' can-say App:isApproved.
'nhs-trust' says 'cacpg' can-say App:A isUsableClinically.
'nhs-trust' says 'mig' can-say App:A isUsableNonClinically.
```

This is all very well, but what apps in practice are approved for use. As the policy document notes, none of the groups or committees have ever approved an app in practice. When we run the satisfiability checker on this policy it reports that (amongst other information) no app is installable.

```
$ java -jar Lint.jar --satisfiability example.policy
[I]: loaded 1/1 files of 6 assertions
Issues identified when checking satisfiability.
The following decisions may be unsatisfiable by their speakers:
'nhs-trust' says * isUsableClinically
'nhs-trust' says * isInstallable
'nhs-trust' says * isApproved
'nhs-trust' says * isUsableNonClinically

In particular the following assertions are unsatisfiable:
'nhs-trust' says App isInstallable if App isApproved, App
    isUsableNonClinically.
'nhs-trust' says App isInstallable if App isApproved, App
    isUsableClinically.

These decisions may be satisfiable through delegation but we
lack any statements to that effect from the delegated party:
(via 'cacpg') 'nhs-trust' says * isUsableClinically
(via 'igc') 'nhs-trust' says * isApproved
(via 'mig') 'nhs-trust' says * isUsableNonClinically
```

As well as reporting which decisions it cannot make, it also reports the specific assertions as well as which decisions could be reached, but rely on delegation, and we have no information from the delegated speakers. This could be because the delegated parties do not believe any app is approved, or it could be because we haven't asked them yet and imported their statements.

This check is somewhat simple and we don't take into account dependencies between variables. If we add, for example, the statements:

```
'igc' says 'angry-birds' isApproved.
'cacpg' says 'dropbox' isUsableClinically.
'mig' says 'instagram' isUsableNonClinically.
```

Then we will still never find any installable apps, as the IGC, CACPG and MIG need to agree on the same app to find it installable. When we run the satisfiability checker however, we find no problems as all the decisions are now satisfiable as there is a decision about *some* variable; even if that variable isn't useful in practice.

```
$ java -jar Lint.jar --satisfiability example.policy
[I]: loaded 1/1 files of 11 assertions
[I]: no completeness problems
```

The satisfiability checker acts as a quick sanity checker that a policy contains enough facts and rules to use all of the rules in the policy; unlike AppPAL proper which can check any specific decision.

### 2.3.2. Checking Redundancy

In logic programming there are two types of redundancy [4]:

- *Unreachability* occurs if a predicate does not take part in the minimal deverivation tree of a fact.
- *Irrelevance* occurs if a derivation tree contains pairs of identical atoms.

These ideas are directly relatable to AppPAL, for instance if we have the following AppPAL policy:

```
'alice' says App isInstallable
  if App isRecommended,
     App isNotMalware.

'alice' says App isRecommended
  if App isNotMalware,
     App isGood.
```

Alice checks that the app is not malware when checking the app is installable and when checking that the app is recommended. The check in `isInstallable` is irrelevant. When writing AppPAL policies this kind of irrelevance commonly occurs when using the typed-syntax described in subsection 2.1. For example, in this excerpt from a SANS BYOD policy there is a check that *U* is a user in both assertions. In the first there is irrelevance because *U* is stated as being a user twice, where once would have been sufficient. In the second there is a single check that *U* is a user but it is irrelevant as the check had already been done when checking is *U* had lost the device (since only users can lose devices in the first assertion).

```
'company' says User:U can-say User:U hasLost(Device:D)
  if D isOwnedBy(U).

'company' says User:U mustInform('help-desk', 'device-lost')
  if U hasLost(Device).
```

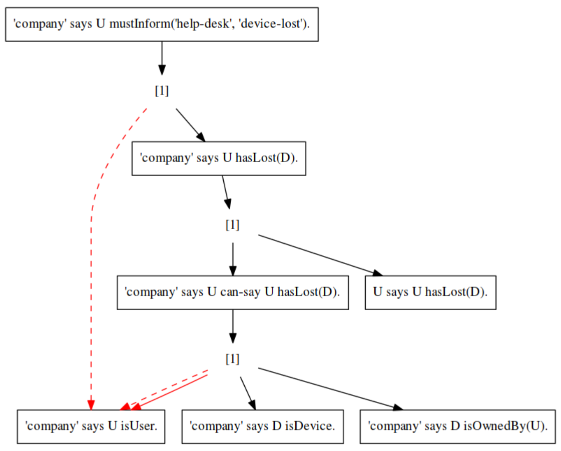

'company' says U mustInform('help-desk', 'device-lost').
[1]
'company' says U hasLost(D).
[1]
'company' says U can-say U hasLost(D).
U says U hasLost(D).
[1]
'company' says U isUser.
'company' says D isDevice.
'company' says D isOwnedBy(U).

Figure 2: Proof graph showing irrelevance.

We can check for this kind of irrelevance by building a proof-graph for the assertion context. Every node (shown in a box) represents an AppPAL fact we might wish to prove. For every assertion in the context we create a proof (represented as a number in brackets) for the head of the assertion, which is connected to the facts required to prove the assertion. In the case of can-say and can-act-as statements we expand them as per AppPAL's inference rules. A proof-graph for the above example is shown in Figure 2: the irrelevant links are shown in red and the AppPAL facts connected to the two dashed ones can be removed to remove the irrelevance.

In contrast unreachability occurs when a fact does not take part in the minimal derivation tree of a fact. As a simple example consider the following policy:

```
'alice' says App isInstallable if App isNotMalware.
'alice' says App isInstallable if App isNotMalware, App isRecommended.
```

To detect unreachability for a given policy we again build the proof-graph (shown in Figure 3). For each proof node we collect the facts (the leaves of the graph underneath it, which are the ground assertions from the AC[3] in AppPAL). If the facts for one proof node connected to a fact are a subset of the facts for another proof node, then the larger proof node, is unreachable it contains facts which are not in the minimal derivation tree. In the case of Figure 3, the derivation-graph 2 is made redundant by derivation-graph 1 as it contains a subset of the facts. This is a simplified example; in general facts lower in the tree may have multiple derivation trees, leading to multiple sets of facts being required for a fact that seems to have only one proof node. Loops can also occur (where one fact depends on itself to prove itself). This approach isn't complete, but it does identify several cases where an AppPAL policy may be redundant through unreachability.

[3]Or in the case of an unsatisfiable policy facts with variables that cannot be unified with a ground fact.

Listing 2: Output of AppPAL when checking a policy with unreachability.

```
java -jar Lint.jar --redundancy unreachable.policy
[I]: loaded 1/1 files of 2 assertions
[I]: flattened 0 node(s)
[W]: 'alice' says App isInstallable. has unreachable derivation trees.
```

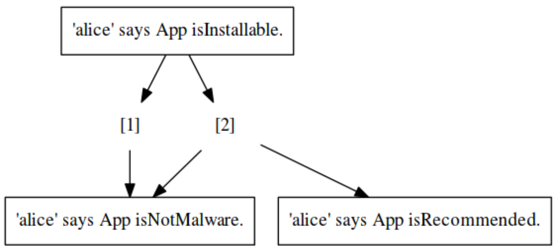

'alice' says App isInstallable.
[1]
[2]
'alice' says App isNotMalware.
'alice' says App isRecommended.

Figure 3: Proof graph showing unreachability.

### Future Directions

I plan to develop at least one more analysis tool for checking consistency. AppPAL does not allow negation in policies, however by using the predicates described in Table 2 a policy author could describe when a subject *can* perform an action as well as when they *cannot*. If the action described is the same then both should not be the same at the same time. Implementing this could be done by searching for examples where both are true, but it would be interesting to see what ground facts that both predicates share (I would expect at least some typing statements) and what the facts they do not share. At least some (or some combination) of those predicates should also never be true at the same time. This may help to create partitions between certain facts and identify which are mutually exclusive even when the names would not trivially identify them.

The redundancy checkers are somewhat buggy, and could be improved. I would also like to write tooling that can check that a policy uses well named predicates, and that everything is documented. In Table 1 I documented what each of the thirty predicates meant. I would like to be able to have a schema that can produce these tables automatically, as well as be able to check that a policy only uses predicates from them. This is technically trivial, but is worthwhile as it helps with documentation and can spot mistakes in policies, such as spelling mistakes in predicates and constants.[4]

In subsection 2.2 I mentioned that I have tools for visualising policies. I would also like to extend these to produce nicer diagrams than the Graphviz diagrams I currently produce which require explanation.

I do not believe there is enough work here for a full paper because there is little novelty

[4]This is especially common when dealing with hyphens in constants. If copied from a PDF document the hyphens are sometimes changed to unicode-dashes which look identical but are counted as a different character. AppPAL supports unicode, so they are parsed correctly but are different symbols. This was an absolute nightmare to debug.

in porting decades old analysis techniques to a variant of an existing language that reduces to Datalog. If this was combined with the BYOD work and I could show that I can detect AppPAL policy problems automatically then it would be interesting, but as it stands the BYOD policies are very simple. Similar ideas have been shown with Datalog and the XACML authorization language [27] (which is well known if overly-complex, poorly specified, and undecidable). I do believe this work is important to continue as it gives AppPAL policies the ability to be more than just another authorization language. It lets AppPAL not only describe policies in mobile ecosystems but potentially identify when those policies are at fault.

## 2.4. Plausible AppPAL

The SecPAL authorization language, and the AppPAL instantiation, allow policy authors to make use of static analysis tools to make decisions, and allow principals to make statements about apps through delegation. When these decisions are made they are made with certainty. If a principal says an app is safe to access the network then we believe that that principal definitely believes the app is safe on a network. When a static analysis tool finds that an app isn't malware then we believe that app to not be malware. This isn't realistic. Static analysis tools can produce false results. A principal might be merely fairly confident that an app can access the network but not absolutely certain.

SecPAL was designed for making access control decisions. The decision whether to install allow a user access to a file or not is a binary one: either they can access it or they cannot. Similarly, the decision process for these decisions is also binary: a user is either logged in or not, a network address is either in the network or outside it, someone can act as someone else's manager or they can not.

AppPAL, however, is primarily for deciding what apps you want to use. Whether you want to install an app or not is less binary than an access control decision, because it is ultimately an opinion. Consider a really simple policy that says *"do not install malware"*: you could try using a malware scanner to check apps, but there opinions on apps can change rapidly and often. You could use a meta-scanning tool like *VirusTotal*, but then you'd only get the percentage of antivirus tools that flagged the app as malicious. Without plausability we cannot represent the doubt and confidence in any assertion.

For another example consider a policy you only want to install apps that are made by *reputable* developers and that are *safe*. Both reputable and safe are poorly defined, and badly represented by a binary decision. A developer may be reputable if they are a large developer with a lot of staff like Google or Facebook, they might be reputable if their apps have been well reviewed, but what about a developer like *King* who produces many games with in-app-purchases, TV adverts, and sketchy privacy records. They are probably more reputable than a malware author but you might have less confidence that they are producing good apps than Google. Similarly, an app might be seen as safe if it doesn't request any permissions and has no native code, but if it starts requesting more permissions and the amount of native code grows then the plausability it is safe should fall. When combined into the policy as a whole Google may be able to get away with a lot more permissions than other developers simply because we trust them more not to be evil. Again, without plausability it is difficult to represent these decisions.

Probabilistic versions of Datalog are not a new idea. Various papers proposed probabilistic variants of Datalog [12] or explored the semantics of probabilistic logics [15]. Role-based access control languages have incorporated ideas about risk into their schemes [19, 10, 28], which is a similar notion to plausability and trust. These schemes do not seem to deal with delegation in the same manner as SecPAL however so incorporating similar ideas here may be interesting and allow SecPAL and AppPAL greater expressiveness. Part of the work toward this this year has included modifying Becker's evaluation and translation-to-Datalog algorithms [7] to include a plausability value, and for giving rules to query on the basis of them, using Datalog$^C$ [20].

### Future Directions

I find the plausability work fascinating, but it is unlikely to progress much further. Adding a confidence to an authorization logic, rather than a more general logic or agent system, seems like a sensible extension and for a limited set of use cases it seems to make sense for AppPAL to be able to reason with false positive rates of static analysis tools, and opinions; however I don't think there is enough here for this work to progress to a full paper or to develop more interesting uses given the time remaining, and I don't believe in contributes greatly to my thesis.

I think it is worth implementing the plausability as part of AppPAL as it helps further distinguish it from SecPAL, and even for a small set of use-cases it is still interesting. Using very simple plausibility combinators will keep it quick to implement (if limited). Implementing and testing can probably be done in less than a week, and a limited demonstration may be a nice addition to my thesis, but extending it to a full paper seems excessive.

## 3. Thesis Progress

My thesis is due by the end of June 2017.

A table of contents for the thesis is shown in Appendix A, and a schedule for completion from January is given in Appendix B. Presently there are just under forty unedited pages written(including table of contents, bibliography, etc); mostly in chapters two and three. Since July, when I returned to fulltime study, I've been aiming to spend a day a week writing the thesis. I expect this time to start increasing to two or three days a week writing after the submission of this report, and for me to be working full-time on it from January. I am planning on having a complete draft finished by April, in time for submission in June.

Next steps include writing the bulk of the thesis, and polish. The literature survey could also do with expansion. Some introductory material has been taken from earlier yearly reports. The sections on users and apps will build on work from my paper at ESSoS [13], and the work in chapter five will build on several unpublished notes from this year.

## References

[1] Requesting Permissions at Run Time | Android Developers.

[2] System Permissions | Android Developers.

[3] Alon Levy, Inderpal Singh Mumick, Yehoshua Sagiv, and Oded Shmueli. Equivalence, query-reachability and satisfiability in Datalog extensions. *PODS*, 1993.

[4] Alon Levy and Yehoshua Sagiv. Constraints and redundancy in datalog. *PODS*, 1992.

[5] A. Armando, R. Carbone, G. Costa, and A. Merlo. Android Permissions Unleashed. In *2015 IEEE 28th Computer Security Foundations Symposium*, pages 320–333, July 2015.

[6] Alessandro Armando, Gabriele Costa, Alessio Merlo, and Luca Verderame. Enabling BYOD through secure meta-market. In *ResearchGate*, August 2014.

[7] Moritz Y. Becker, Cdric Fournet, and Andrew D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, January 2010. 00059.

[8] Avik Chaudhuri. Language-based security on {A}ndroid. In *ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security*, pages 1–7, New York, 2009. ACM. Cited by 0000.

[9] G. Costantino, F. Martinelli, A. Saracino, and D. Sgandurra. Towards enforcing on-the-fly policies in BYOD environments. In *2013 9th International Conference on Information Assurance and Security (IAS)*, pages 61–65, December 2013.

[10] Nathan Dimmock, Andrs Belokosztolszki, David Eyers, Jean Bacon, and Ken Moody. Using Trust and Risk in Role-based Access Control Policies. In *Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies*, SACMAT '04, pages 156–162, New York, NY, USA, 2004. ACM.

[11] 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.

[12] Norbert Fuhr. Probabilistic Dataloga logic for powerful retrieval methods. *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, 1995.

[13] Joseph Hallett and David Aspinall. AppPAL for Android. In Juan Caballero, Eric Bodden, and Elias Athanasopoulos, editors, *Engineering Secure Software and Systems*, number 9639 in Lecture Notes in Computer Science, pages 216–232. Springer International Publishing, April 2016. DOI: 10.1007/978-3-319-30806-7_14.

[14] Joseph Hallett and David Aspinall. Specifying BYOD Policies with Authorization Logic. In *PhD Symposium at iFM'16 on Formal Methods*, Reykjavik University, June 2016. Reykjavik University.

[15] Joseph Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3):311–350, December 1990.

[16] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 639–652, New York, NY, USA, 2011. ACM. Cited by 0024.

[17] https://www.statista.com/statistics/236027/global-smartphone-os-market-share-of-android/. Android OS smartphone shipments market share 2011-2016 | Statistic.

[18] 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*, pages 3–14, 2012. Cited by 0012.

[19] Audun Jsang and Stphane Lo Presti. Analysing the Relationship between Risk and Trust. In Christian Jensen, Stefan Poslad, and Theo Dimitrakos, editors, *Trust Management*, number 2995 in Lecture Notes in Computer Science, pages 135–145. Springer Berlin Heidelberg, March 2004. DOI: 10.1007/978-3-540-24747-0_11.

[20] Ninghui Li and John C. Mitchell. Datalog with Constraints: A Foundation for Trust Management Languages. In Veronica Dahl and Philip Wadler, editors, *Practical Aspects of Declarative Languages*, number 2562 in Lecture Notes in Computer Science, pages 58–73. Springer Berlin Heidelberg, January 2003. DOI: 10.1007/3-540-36388-2_6.

[21] Jialiu Lin, Bin Liu, Norman Sadeh, and Jason I. Hong. Modeling Users Mobile App Privacy Preferences: Restoring Usability in a Sea of Permission Settings. *Symposium On Usable Privacy and Security*, pages 199–212, 2014.

[22] Fabio Martinelli, Paolo Mori, and Andrea Saracino. Enhancing Android Permission Through Usage Control: A BYOD Use-case. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, pages 2049–2056, New York, NY, USA, 2016. ACM.

[23] Abadi Martn. Logic in Access Control. 2003.

[24] MobileIron Security Labs. Q4 Mobile Security and Risk Review. Technical report, MobileIron Security Labs, December 2015.

[25] Jennifer E. Moyer. Managing Mobile Devices in Hospitals: A Literature Review of BYOD Policies and Usage. *Journal of Hospital Librarianship*, 13(3):197–208, July 2013.

[26] Rikesh K. Patel, Adele E. Sayers, Nina L. Patrick, Kaylie Hughes, Jonathan Armitage, and Iain Andrew Hunter. A UK perspective on smartphone use amongst doctors within the surgical profession. *Annals of Medicine and Surgery*, 4(2):107–112, June 2015.

[27] Carroline Dewi Puspa Kencana Ramli. Detecting Incompleteness, Conflicting and Unreachability XACML Policies using Answer Set Programming. *arXiv:1503.02732 [cs]*, March 2015. arXiv: 1503.02732.

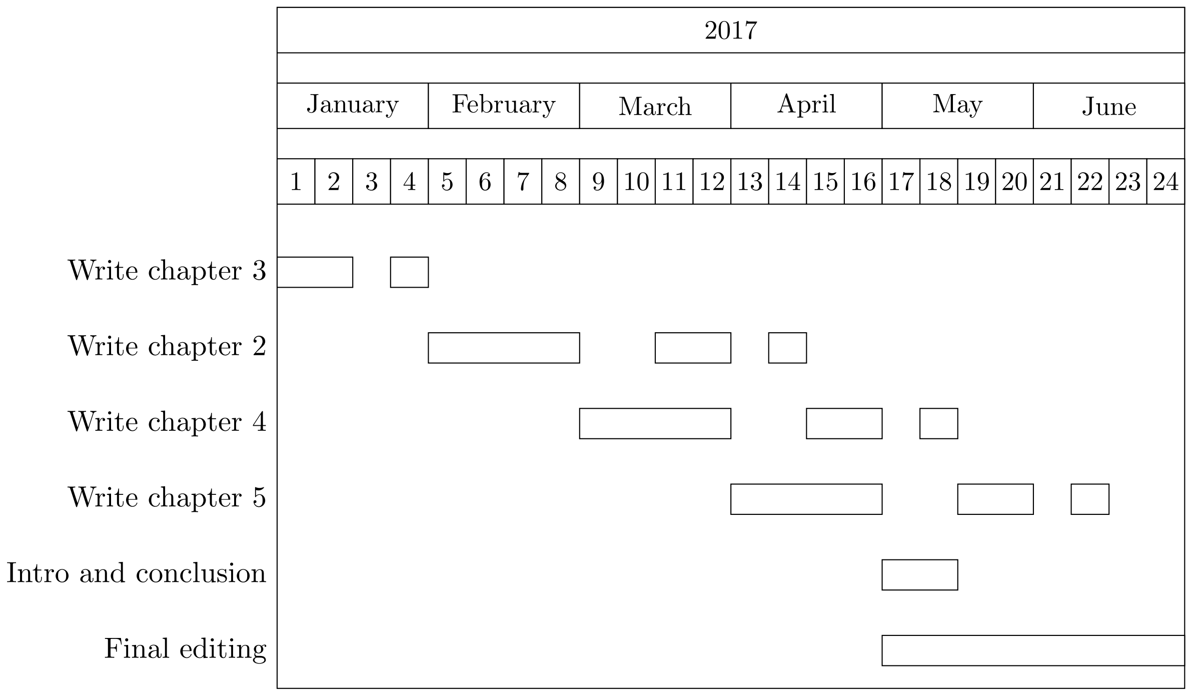
[28] F. Salim, J. Reid, E. Dawson, and U. Dulleck. An Approach to Access Control under Uncertainty. In *2011 Sixth International Conference on Availability, Reliability and Security (ARES)*, pages 1–8, August 2011.

[29] Holger Schulze. BYOD & Mobile Security 2016 Spotlight Report. Technical report, LinkedIn Information Security, 2016.

# A. Thesis Outline

1. Introduction
2. Background
   - 2.1 Android and mobile OSs
     - 2.1.1 Overview of architecture
     - 2.2.2 Changes to Android
     - 2.3.3 Overview of mobile devices and functionality
     - 2.4.4 Mobile Ecosystems
   - 2.2 Authorization logics and Fine Grained Permission Systems
     - 2.3.1 PolicyMaker and Keynote
     - 2.3.2 SPKI/SDSI
     - 2.3.3 RT
     - 2.3.4 SecPAL and DKAL
     - 2.3.5 Cassandra
     - 2.3.6 XACML
     - 2.3.7 Dr. Android and Mr. Hide
     - 2.3.8 Aurasium
     - 2.3.9 CRePE
     - 2.3.10 Kirin
     - 2.3.11 SEAndroid
   - 2.3 SecPAL
3. AppPAL
   - 3.1 Why SecPAL
   - 3.2 Instantiating SecPAL for mobile ecosystems
     - 3.2.1 Changes to SecPAL
     - 3.2.2 Typed AppPAL
     - 3.2.3 Predicate conventions
   - 3.3 Examples of AppPAL
   - 3.4 Implementation
4. Apps and App Stores
   - 4.1 Users and Apps
     - 4.1.1 What privacy preferences do users have?

## B. Schedule for Completion



Shown above is a Gantt chart for how I expect to spend my time from January writing the thesis. I have split the writing into three stages: first draft, first edit second edit, each with a gap between to let me collect feedback.

I am aiming to have a near-complete first draft ready by the start of May, and the final completed thesis by the end of June (when my funding ends). As mentioned in section 3 some work has started in most chapters, and I intend to get more done by the start of January. This schedule is, essentially, a worst case schedule. Ideally work on some chapters should commence earlier than planned, and it may not be as linear as prescribed here and some chapters have already been started.

I have also kept January and February relatively quiet as I will have a newborn baby to look after, and I don't know how this will affect my work. Assuming it isn't doesn't slow me down too much then I can always press on with other writing and move the start dates back. If it does slow my work down, however, then at least I have planned for it. From my perspective it is better to have a strict schedule about when things need to be done (or at least be being reviewed) than give overly optimistic deadlines and miss them.

## C. Doctoral Symposium Paper from iFM

# Specifying BYOD Policies With Authorisation Logic

Joseph Hallett and David Aspinall

University of Edinburgh

**Abstract**

BYOD policies are imprecisely specified using natural language. This creates ambiguity, may hide specification problems, and makes comparison harder. We show how we can use AppPAL (an authorisation logic) to formalise three BYOD policies. We identify potential problems in all three policies. This suggests authorisation logic may help in policy specification and enforcement.

## 1 Introduction

Employees bring their personal devices to work. Corporate IT departments can find this challenging: they want to secure their networks but have limited control over a personal device's software and configuration. To ensure compliance IT departments publish mobile device policies. Employees must agree to follow them if they want to use their personal devices at work. Policies specified using natural language can contain ambiguities. This makes compliance checking hard as automatic methods are hard to use with these policies.

Logics of authorisation are languages used to express permissible actions. Their uses include access control [6], and identifying problems in electronic health records [8] amongst other applications. *AppPAL* [3] instantiates Becker et al.'s SecPAL [1] language to express mobile app installation policies. It makes use of delegation relationships and uses external constraints to formalise policies. AppPAL policies define predicates which can affect application installation (*isInstallable*) or operation; they may be enforced by application rewriting or a modified version of the mobile platform security manager. We believe AppPAL has applications to mobile device BYOD policies as well.

AppPAL is designed to be readable; we introduce the language alongside example excerpts from three BYOD including: a security policy from the SANS institute to aid companies in developing their own rules [2], an NHS hospital trust [4], and a policy used at The University of Edinburgh [5]. We show how AppPAL can specify BYOD policies making them precise, identifying problems and aiding comparison.

## 2 Translating Corporate BYOD Policies

Many BYOD policies describe the apps usable in the workplace. The Edinburgh policy recommends apps come from reputable sources (though the meaning of *reputable* is left undefined). The SANS policy encourages the IT department to maintain black-lists and white-lists. In fact the SANS policy recommends this multiple times in their document.

**Edinburgh**: *"Only download applications (apps) or other software from reputable sources."*

```
'edinburgh−policy' says Store can-say App isInstallable if Store sells(App), Store isReputable.
```

**SANS**: *"The IT Department maintains a list of allowed and unauthorised applications and makes them available to users on the intranet."*

```
'sans−policy' says 'it−department' can-say App isInstallable.
```

**SANS**: *"Only approved third party applications can be installed on handhelds. The approved list can be obtained by contacting the IT department, or should be available on the intranet."*

```
'sans-policy' says 'it-department' can-say App isInstallable.
```

The NHS policy distinguishes between *clinical* and *non-clinical* apps; with both requiring approval through a different group before a committee gives final approval. In practice these rules are irrelevant: no group has ever approved an app. This is, perhaps, unsurprising as 82% of Android medical apps have basic privacy concerns [7, 9].

**NHS**: *"Apps for work usage must not be downloaded onto corporately issued mobile devices (even if approved on the NHS apps store) unless they have been approved through the following Trust channels: Clinical apps; at the time of writing there are no apps clinically approved by the Trust for use with patients/clients. However, if a member of staff believes that there are clinical apps [· · ·] ratification should be sought via the Care and Clinical Policies Group. [· · ·] Business apps; at the time of writing there are no business (i.e., non-clinical) apps approved by the Trust for use other than those preloaded onto the device at the point of issue. However, if a member of staff believes that there are apps [· · ·] ratification of the app must be sought via the Management of Information Group (MIG). [· · ·] Following approval through Care and Clinical Policies and/or MIG, final approval will be required through Integrated Governance Committee."*

```
'nhs-policy' says App isInstallable if App isUsable, App isFinallyApproved.
'nhs-policy' says 'cacpg' can-say App isUsable if App isUsableClinically.
'nhs-policy' says 'mig' can-say App isUsable if App isUsableNonClinically.
'nhs-policy' says 'igc' can-say App isFinallyApproved.
```

All three policies use delegation to decide is an app is installable. By translating the policies into AppPAL problems become apparent: the Edinburgh policy is *incomplete* as it lacks a means to determine what stores were reputable, the SANS policy *contains duplicated rules*, and the NHS policy is incomplete and contains *redundant statements* as no app has ever been approved for use. For all three policies the AppPAL rule is no longer than the natural language policy, and in the case of the NHS policy considerably shorter and clearer.

Similarities appear when looking at how devices connect to networks. The Edinburgh policy recommends disabling automatic access to open unsecured Wi-Fi networks (though again it does not specify what *secure* actually means). In contrast, the SANS policy says what security features are required and that, whilst secure access points are preferred, an open access point is acceptable if used with a VPN.

**Edinburgh**: *"Control your devices connections by disabling automatic connection to open, unsecured Wi-Fi networks."*

```
'edinburgh-policy' says Device canConnectTo(AP) if AP isSecure.
```

**SANS**: *"If mobile workers do require connectivity through public, open, or untrusted WLAN, then users MUST use WLANs using, if available and in this order: WPA(2) encryption, WEP 256 bits (or 128 bits), or finally open networks if nothing else is available. Users connected to data networks in an open environment MUST use a VPN connection."*

```
'sans-policy' says Device canConnectTo(AP) if AP isSecure.
'sans-policy' says AP isSecure if AP canUseWPA2.
'sans-policy' says AP isSecure if AP canUseWEP256.
'sans-policy' says AP isSecure if AP canUseWEP128.
'sans-policy' says Device canConnectTo(AP) if Device hasVPN(VPN) where connect(VPN) = true.
```

By analysing the similarities in policies a framework of BYOD predicates could be defined that would allow policies to be compared using standard terms *precisely*.

AppPAL can describe complex scenarios in policies. The NHS policy describes when a clinician can use their phone's camera as part of patient care. The rules require obtaining consent (delegation to the patient) as well as clinical approval (delegation to the clinician).

As well as helping clarify the policy, the AppPAL rules can aid with compliance—a proof an AppPAL policy was satisfied might act as evidence that proper procedure was followed.

**NHS**: *"Some mobile devices have the ability to take photographs/videos. This function should not be used for photographs/videos of an individuals care and treatment unless the device has encryption enabled and it is clinically appropriate to do so. If the photography/video facility is used as part of the recording of an individuals care and treatment, the device user must ensure that the consent of the individual has been collected prior to taking any photograph/video.* [· · ·] *A record of the consent must be entered into the individuals care record."*

```
'nhs—policy' says Device canPhotograph(Patient) if Device isEncrypted, Patient canBePhotographed.
'nhs—policy' says 'clinician' can-say Patient canBePhotographed
  if Patient hasClinicalNeedForPhotograph, Patient hasConsentedToPhotograph
  where recordedICRConsentForPhotograph(Patient) = true.
'nhs—policy' says 'clinician' can-say Patient hasClinicalNeedForPhotograph.
'nhs—policy' says Patient can-say Patient hasConsentedToPhotograph if Patient canConsent.
```

## 3 Current Status

So far we have developed AppPAL policies describing each of the BYOD policies. We have developed tools to identify when AppPAL policies may be incomplete, and aim to extend them to inconsistency and redundancy checks in future work. For example, when checking the *isInstallable* predicate in the NHS policy our completeness checker spots the missing statements from the delegated to groups, and that the *isInstallable* rule is unsatisfiable.

```
The following assertions are unsatisfiable:
  'nhs—policy' says App.1 isInstallable if App.1 isUsable, App.1 isFinallyApproved.
These decisions may be derivable but we lack statements from the delegated party:
  (via 'mig') 'nhs—policy' says * isUsable
  (via 'igc') 'nhs—policy' says * isFinallyApproved
  (via 'cacpg') 'nhs—policy' says * isUsable
```

As the BYOD policies are informal we should question the accuracy of our formalisation. Future work will look at developing a testing process as part of the requirements capture, and working with policy owners to check that rules AppPAL postulates as true indeed match the author's intentions. In general, BYOD policies are at an early stage. As policies grow and become more intricate, we believe that tools like AppPAL will help ensure they are well specified.

## References

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

[2] N.R.C. Guérin. Security Policy for the use of handheld devices in corporate environments, *SANS*.

[3] J. Hallett and D. Aspinall. AppPAL for Android. *Engineeering Secure Software and Systems*, 2016.

[4] G. Kennington, et al. Mobile Devices Policy, *Torbay and Southern Devon Health Care NHS Trust.*

[5] D. Williamson, A. Grzybowski, and S. Graham. BYOD Policy: Use of Personally Owned Devices for University Work, *University of Edinburgh.*

[6] M. Abadi. Logic in access control, *Logic in Computer Science*, 2003.

[7] S.R. Blenner, et al. Privacy Policies of Android Diabetes Apps and Sharing of Health Information *Journal of the American Medical Association*, 2016.

[8] M.Y. Becker, and P. Sewell, Cassandra: flexible trust management, applied to electronic health records, *Computer Security Foundations*, 2004.

[9] K. Knorr, and D. Aspinall. Security Testing for Android mHealth Apps *Software Testing, Verification and Validation Workshops*, 2015.