Authorization Logic for Mobile Ecosystems

Third Year Report

Joseph Hallett

January 24, 2017

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

⟨E⟩ ≔ ⟨Variable⟩ | **'constant'**
⟨Variable⟩ ≔ **(Type:)?VariableName**

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 (extensional 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).
```

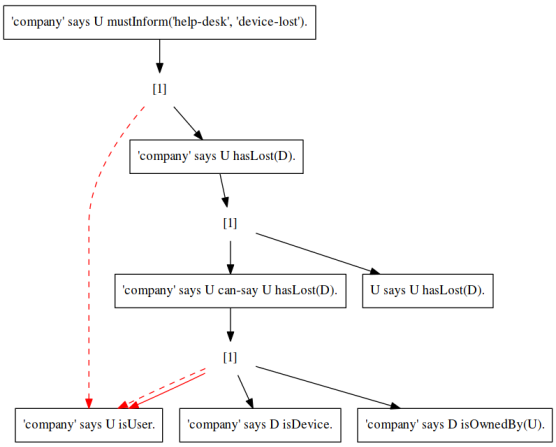


Figure 2: Proof graph showing irrelevance.

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.
```

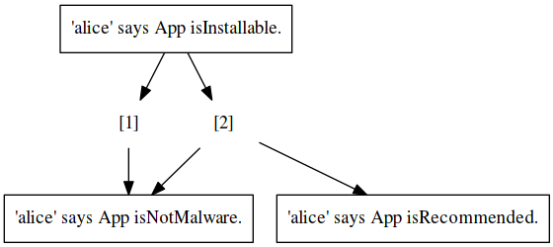


Figure 3: Proof graph showing unreachability.

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

The plausability work 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.

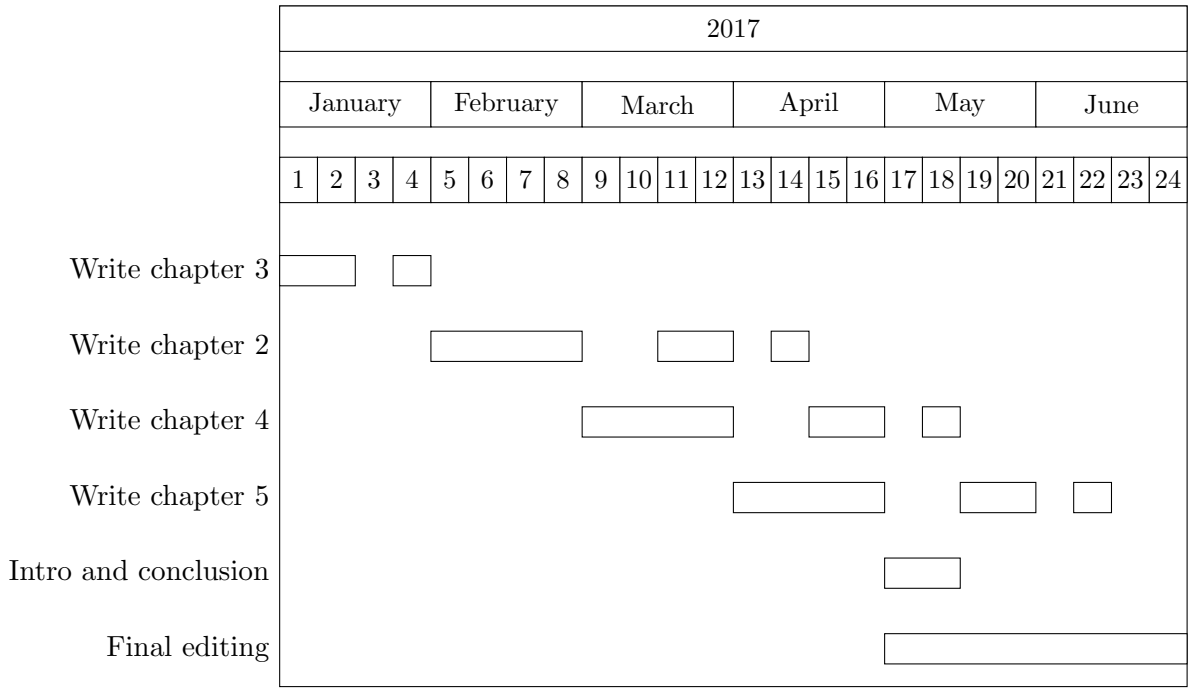
[28] F. Salim, J. Reid, E. Dawson, and U. Dulleck. An Approach to Access Control under Uncertainty. In *2011 Sixth International Conference on Availability, Reliability and Security (ARES)*, pages 1–8, August 2011.

[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
   - 3.5 Inferring problems with AppPAL policies automatically
     - 3.4.1 Satisfiability
     - 3.4.2 Redundancy

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.* [$\cdots$] *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.

D. Short Paper Submission to IMPS 2017

Common Concerns in BYOD Policies

Joseph Hallett
University of Edinburgh

David Aspinall
University of Edinburgh

***Abstract*—Companies publish BYOD policies for employees to use their device at work. These policies are written using natural language, and vary in length and style. Using BYOD policies from five different sources we explore common concerns. Existing tools for enforcing policies has focussed on restricting app and device functionality. Our work looks at 5 BYOD policies and presents the common concerns and structures found in the policies themselves. This suggests where future MDM tools should focus their efforts.**

I. Introduction

Many employees bring their personal mobile devices to work. To control the access these devices have, 70% of companies publish *bring your own device* (BYOD) policies [16]. These BYOD policies are written documents for employees to read and follow. They describe steps to take to secure devices appropriate to the workplace. The policies say how employees should access data, and who should authorise decisions.

Companies have a variety of means to implement their policies. Some companies may trust employees to follow the rules on their own. Alternatively *Mobile Device Management* (MDM) software can implement part of the policies: packages such as IBM's MaaS360 and Blackberry's BES [9, 17] can configure devices to restrict functionality and manage apps. Research has looked at developing other tools such as UC-Droid [12] or BYODroid [2] which was used to implement parts of a NATO BYOD policy [1].

Commercial tools are limited in what polices they can enforce. Some tools can only enable simple on-off configuration settings, and ban explicitly black-listed apps. More sophisticated systems can use app-rewriting to recompile apps to tunnel traffic through a VPN, or geofencing to apply policies in predefined areas. These tools are not infallible. One survey found that 50% of companies with MDM software still had non-compliant devices in their networks [13]. Whilst app wrapping can protect some apps, in general it is ineffective [7].

MDM software and research has, so far, focused on restricting apps and device functionality. But what are the concerns and restrictions described in the policies themselves? By analysing 5 BYOD policies, we present the common concerns and structures found in the policies themselves. The policies display various styles for writing policies, and differing concerns. In comparison to MDM software, the policies do not focus exclusively on restrictions but rather require employees acknowledge company rules and the policies describe trust relationships amongst different departments. This suggests little consensus on the best way to write such policies and that more research is needed.

II. BYOD Policies

We analysed 5 policies from different sources to find common concerns. These five were chosen as they come from a variety of sources and industries. Two are advisory, published specialist institutions to help other companies implement their own policies. Three are policies used in a hospital, a company selling emergency sirens, and a university.

- The SANS policy [14] is a hypothetical BYOD policy that a company could use as a starting point to base its own policy on. It is prescriptive and long, focussed on technical restrictions such as disabling device features.
- The HiMSS policy [8] also is a hypothetical policy, for US hospital trusts planing to implement a BYOD policy. It is short, but and written from the perspective of the employee agreeing to the policy. This is different to the other policies where instead the policy is written as rules for employees to follow.
- The NHS policy [10] is a BYOD policy used in a British hospital trust. It is long, and describes a complex policy in a large organisation with a complex hierarchy.
- The final two policies are simpler, the fourth is taken from a company selling emergency sirens for cars [4], the fifth is by the University of Edinburgh. Both are short, relatively uncomplicated, but typical examples of policies found *in the wild*.

Not all the policies are written in the same style. Most are written from the perspective of a company or IT department telling an employee what to do. The HiMSS policy, however, is written as a contract where the user tells the company how they will behave. Most policies separate individual policy rules into small individual rules each which must be followed. The Edinburgh policy groups them together into two or three large rules, with different increasing sets of sub-rules for low and high risk employees to follow.

III. Translation to SecPAL

To help compare the policies, they were first translated them into a formal language [6]. We use a dialect of SecPAL [3] that we used for describing app privacy preferences [5]. SecPAL is designed to be readable, and requires an explicit authority to *speak* individual statements. This allows SecPAL-based languages to capture delegation relationships and the differences in style between policies. Understanding the contents of policies and making comparisons is easier when using SecPAL as it helps to remove the ambiguities of natural language [6].

Each of the policies are split into a series of rules. The SecPAL used to translate the rules has a structure similar to

Datalog. An authority (the speaker) will decide that a fact is true, if it can be convinced of a series of conditional facts are also true.

⟨speaker⟩ says fact⟨X⟩ ⟨predicate⟩ if condition⟨Y⟩ ⟨predicate⟩ · · · .

A simple example is the following example from the Sirens-company policy. The policy states that devices may access various company resources. For each resource we create a SecPAL assertion that states that a device can access it.

Sirens: *Employees may use their mobile device to access the following company-owned resources:*
● *Email* ● *Calendars* ● *Contacts* ● *Documents* ● *Etc.*

```
'department' says Device:D canAccess('email').
'department' says Device:D canAccess('calendars').
'department' says Device:D canAccess('contacts').
'department' says Device:D canAccess('documents').
```

A more complex example can be taken from the NHS policy. Employees are not allowed to call non-domestic, or premium rate numbers on company-owned phones, however an exception can be made if approved by the appropriate manager. To implement this we have the default rule that international calls are banned, a second rule stating that it is allowed if an exemption is made, finally a third rule delegating the exemption making process to the employee's manager.

NHS: *All mobile devices will be configured for national access only. Premium/international calls will be barred. International call barring and roaming arrangements can be lifted for specific periods, to be stipulated on request, on approval of the relevant manager/budget holder.*

```
'nhs-trust' says Device canCall(TelephoneNumber:X)
 if Device isOwnedBy('nhs-trust'),
   X isNationalNumber, X isStandardRateNumber.

'nhs-trust' says Device canCall(TelephoneNumber:X)
 if Device isOwnedBy(Staff),
   Staff hasCallExemption.

'nhs-trust' says Manager can-say
 Staff hasCallExemption
 if Manager isManagerOf(Staff).
```

The formalisation of the policies[1] and tooling for our variant of SecPAL[2] are available online.

IV. Rule Structures

The predicates used in the formalisation of the rules fall into 4 categories. *Can* predicates describe what their subjects can do; for instance whether a device can connect to a server. *Must* predicates describe obligations, such as reporting a lost device. *Has* predicates ensure an action has been completed in the past, such as approving an app. Finally, *is* predicates describe a typing property about their subjects.

The occurrence of each type of predicate is shown in Table I. The use of each is also split by whether the predicate is a *decision* made by the policy, or a *condition* for making that decision. *Can* and *must* decisions feature in all policies excepting *can* decisions in the Edinburgh policy, in part due to the structure of the policy as discussed in section II. This is expected, these are access control decisions and reactions to events; both topics that existing MDM tools have focused on implementing. *Has* and *is* predicates are the majority of the conditions, but there are also decisions using them too.

[1]https://github.com/apppal/apppal-byod-policy-translations
[2]https://github.com/apppal/libapppal

	Decision				Condition			
Policy	Can	Must	Has	Is	Can	Must	Has	Is
SANS	35% (35)	29% (29)	9 % (9)	27% (27)	2% (2)	2% (2)	8% (8)	87% (87)
HiMSS	21% (21)	41% (41)	31% (31)	7 % (7)	0	0	13% (13)	87% (87)
NHS	19% (19)	26% (26)	33% (33)	23% (23)	2% (2)	0	19% (19)	83% (83)
Sirens	27% (27)	45% (45)	11% (11)	16% (16)	2% (2)	7% (7)	2% (2)	89% (89)
Edinburgh	0	18% (18)	82% (82)	0	7% (7)	7% (7)	50% (50)	37% (37)

TABLE I
Occurrences of predicate-types in each policy.

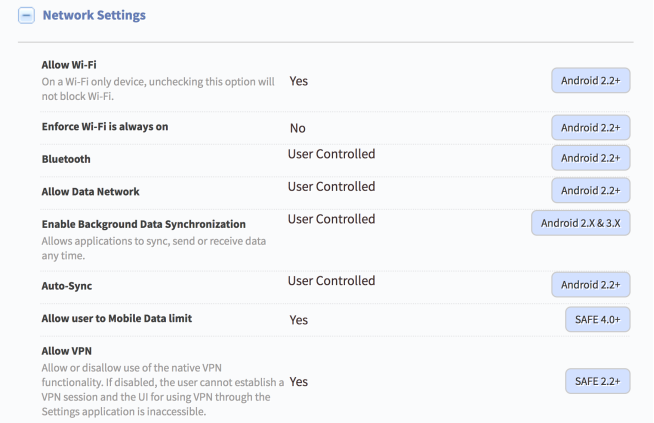


Fig. 1. Example policy from the MaaS360 MDM software.

Existing MDM tools present policies as a series of tick-boxes for what a device *can* and *must* do (Figure 1). An administrator selects which policies each device must follow, a predominantly manual process. In our examination of the policies, as well as finding rules that describe what a device *can* do, we find rules that group devices by what they *have* or *are*. Selecting which restrictions to apply to a device is defined by policies; but existing MDM tools do not allow policies to be selected on the basis of policies. MDM tools perhaps need greater flexibility to fully implement all aspects of a BYOD policy.

V. Common Concerns and Checks

Analysing each of the policies common concerns become apparent. A summary of predicates, with the same meaning, used in multiple policies by our translation is given in Table II.

Acknowledgements, where individuals are asked to acknowledge other policies, and predicates linking devices to owners are used in all policies. Most policies described rules for when device features should be enabled and disabled. Configuring device features is a common feature to many MDM packages, but tracking what a user agrees to is not seen in leading MDM packages like MaaS360 or BES [15]. Only 2 out of 5

Predicate	SANS	HiMSS	NHS	Sirens	Edinburgh
mustAcknowledged	✓	✓	✓	✓	✓
hasAcknowledged	✓	✓	✓	✓	✓
isOwnedBy	✓	✓	✓	✓	✓
isDevice	✓	✓	✓	✓	✓
mustDisable	✓		✓	✓	✓
isLost	✓	✓	✓	✓	
isEmployee	✓		✓	✓	✓
isApp	✓	✓	✓	✓	
isActivated	✓	✓	✓		✓
mustEnable	✓	✓		✓	
mustWipe		✓	✓	✓	
isEncrypted	✓		✓		✓
hasMet	✓		✓		✓
canMonitor	✓		✓	✓	
mustInform	✓		✓		
isTelephoneNumber	✓		✓		
isString			✓	✓	
isSecurityLevel	✓	✓			
isInstallable	✓		✓		
isFeature	✓			✓	
isData	✓			✓	
isApprovedFor		✓	✓		
isApproved	✓	✓			
hasFeature			✓		✓
hasDevice		✓	✓		
hasDepartment		✓			✓
canUse	✓		✓		
canStore	✓	✓			
canInstall	✓		✓		
canConnectToServer	✓		✓		
canConnectToNetwork	✓			✓	
canConnectToAP	✓	✓			
canCall	✓		✓		
canBackupTo		✓			✓

TABLE II
OCCURRENCES OF PREDICATES COMMON TO MULTIPLE POLICIES.

policies had rules limiting what networks, servers, or access points a device could access; and only the two most complex policies had rules limiting what apps could be installed. This is surprising as a common feature of MDM tools is controlling how devices and apps access networks. Users have privacy preferences about apps [11], but not all companies try to control what apps employees install. Providing curated app stores and blacklisting apps is a feature common to many MDM programs. Not all policies express rules about which apps to install, however.

All the policies we looked at required employees to be aware of and acknowledge the existence of other policies. The use of acknowledgements is noteworthy because policies acknowledged may not be ones that are enforcible automatically. These other rules include ethical or legal guidelines and disclaimers about data-loss. Writing software to check a user is aware they may lose the data, and is behaving ethically may not be possible.

An aspect of acknowledgements is that they require a delegation of trust from the company to their employees. The employees have to be responsible for stating what they do or do not acknowledge. Delegation is key to other policy decisions in the policies. The IT department is delegated to audit apps. Users are responsible for reporting their device missing.

	SANS	HiMSS	NHS	Sirens	Edinburgh
Rules in policy	33	15	56	20	25
Rules using acknowledgement	6% (6)	67% (67)	20% (20)	24% (24)	5% (5)
Rules using delegation	70% (70)	33% (33)	59% (59)	52% (52)	10% (10)
Rules describing a restriction	54% (54)	20% (20)	14% (14)	20% (20)	5% (5)

TABLE III
SUMMARY OF IDIOMS IN EACH OF THE POLICIES.

We summarise the number (and percentage) of rules in the policies using acknowledgements and delegation relationships in Table III. For comparison, we also give the at the number of rules which describe some form of restriction. Delegation relationships form a significant proportion of all the policies. Acknowledgements are used extensively in the HiMSS policy, but rarely in the SANS policy. Overall acknowledgements form as much a part of the policies as do device restrictions. MDM software has focussed on implementing device restrictions and configurations; but it would seem that other aspects are equally important.

VI. AUTHORITIES AND DELEGATION

Each of the policies use delegation to describe rules. A delegation requires at least two parties: someone to hand off the decision, and someone to hand the decision to. They can be individuals, but often they are a role that several individuals may fulfil When translating the policies into SecPAL we created an authority to deliver the policy. In most cases we took the company (who had authored the document) as the authority. In the HiMSS policy, however, rules are phrased as a user stating what they will do.

Most policies used three authorities to make the bulk of the decisions. A primary authority expresses the bulk of the policy and delegation relationships. They act as the *voice* of the policy. The primary authority delegates to the technical authority for some decisions. They may maintain inventories of devices, approve apps for devices, and describe what users may connect to. Their job varies between policies, but in general they are delegated to in order to provide more detailed policy rules. The user is responsible for stating who they are, what devices they have, and what the status of their device is (is it lost or no longer required, for instance).

Some policies have more authorities, than others (Table IV). The NHS policy has various managers that approve decisions for their staff. There are different groups that make decisions for the clinical and business halves of the business. If a clinical user wishes to use an app with a patient they must seek approval from two policy groups, as well as their line manager. Others make less use of different authorities. In the Edinburgh policy, the records-management office states how a low or high risk

SANS	Authorities	10
	Primary Authority	company
	Technical Authority	it-department
	User Authority	user
HiMSS	Authorities	3
	Primary Authority	user
	Technical Authority	xyz-health-system
	User Authority	department
NHS	Authorities	11
	Primary Authority	nhs-trust
	Technical Authority	it-department
	User Authority	staff
Sirens	Authorities	4
	Primary Authority	department
	Technical Authority	it-department
	User Authority	employee
Edinburgh	Authorities	2
	Primary Authority	records-management
	Technical Authority	
	User Authority	employee

TABLE IV
SUMMARY OF DIFFERENT AUTHORITIES IN POLICIES.

must be configured. There is no delegation to others to further specify aspects of the policy. Delegation of responsibilities is an important part of BYOD policies. MDM software seems largely to ignore it, however. These tools instead allow IT staff to set fixed policies and push them to devices. No further requesting of information is typically needed or required.

VII. CONCLUSIONS

We have looked at 5 different BYOD policies and presented a summary of their contents, their styles of presentation and the relationships within them. To analyse the policies we translated them into SecPAL. The translation from natural to formal language is somewhat subjective. We have tried to mitigate this by attempting to preserve the style of the original and by careful use of predicates between different policies. Working with a company to implement the SecPAL policy inside their business would help to ensure the formalisation is correct.

Comparing the policies we found a diverse range of concerns. Some of these concerns (the use of acknowledgements in particular) are not addressed by current BYOD tools. The tools focus on device configuration, which are only part of the concerns of the BYOD policies. We also found the policies were presented in a variety of styles. Some, like SANS, are prescriptive and describe a company telling employees what to do. Others, like HiMSS, take the form of a user contract, where employees state that they acknowledge the companies rules and agree to follow them. The lack of commonality suggest there may be no perfect solution for implementing BYOD policies. Future tools must be flexible enough to express a range of policies, and model the trust relationships these policies contain.

REFERENCES

[1] A. Armando et al. "Developing a NATO BYOD security policy". In: *International Conference on Military Communications and Information Systems*. May 2016.

[2] A. Armando et al. "Enabling BYOD through secure meta-market". In: Aug. 2014.

[3] M. Y. Becker, C. Fournet, and A. D. Gordon. "SecPAL: Design and semantics of a decentralized authorization language". In: *Journal of Computer Security* (Jan. 2010).

[4] Code3PSE.org. *Sample BYOD Policy*. URL: http://www.code3pse.com/public/media/22845.pdf (visited on 10/14/2016).

[5] J. Hallett and D. Aspinall. "AppPAL for Android". In: *Engineering Secure Software and Systems*. Apr. 2016.

[6] J. Hallett and D. Aspinall. "Specifying BYOD Policies with Authorization Logic". In: *PhD Symposium at iFM'16 on Formal Methods*. Reykjavik University: Reykjavik University, June 2016.

[7] H. Hao, V. Singh, and W. Du. "On the effectiveness of API-level access control using bytecode rewriting in Android". In: *ASIA CCS* (2013).

[8] Healthcare Information and Management Systems Society. "Mobile Security Toolkit: Sample Mobile Device User Agreement". In: *Healthcare Information and Management Systems Society* (2012).

[9] *IBM MaaS360 - Enterprise Mobility Management (EMM)*. URL: http://www-03.ibm.com/security/mobile/maas360.html (visited on 10/12/2016).

[10] G. Kennington et al. *Mobiles Devices Policy*. Tech. rep. Torbay, Southern Devon Health, and Care NHS Trust, Mar. 2014.

[11] J. Lin et al. "Modeling Users Mobile App Privacy Preferences: Restoring Usability in a Sea of Permission Settings". In: *Symposium On Usable Privacy and Security* (2014).

[12] F. Martinelli, P. Mori, and A. Saracino. "Enhancing Android Permission Through Usage Control: A BYOD Use-case". In: *Symposium on Applied Computing*. 2016.

[13] MobileIron Security Labs. *Q4 Mobile Security and Risk Review*. Tech. rep. MobileIron Security Labs, Dec. 2015.

[14] Nicholas R. C. Guerin. *Security Policy for the use of handheld devices in corporate environments*. Tech. rep. SANS, May 2008.

[15] Rob Smith et al. *Magic Quadrant for Enterprise Mobility Management Suites*. Tech. rep. G00279887. Gartner, June 2016. URL: https://www.gartner.com/doc/reprints?id=1-390IMNG&ct=160608&st=sb (visited on 11/23/2016).

[16] H. Schulze. *BYOD & Mobile Security 2016 Spotlight Report*. Tech. rep. LinkedIn Information Security, 2016.

[17] *Secure Android Solution BlackBerry and Android for Work*. URL: http://us.blackberry.com/enterprise/android-for-work.html (visited on 12/13/2016).

E. Paper Submission to IFIP-SEC 2017

Capturing Policies for BYOD

Joseph Hallett and David Aspinall

School of Informatics, University of Edinburgh

Abstract. BYOD policies are informally specified using natural language. We show how the SP4BYOD language can help reduce ambiguity in 5 BYOD policies and link the specification of a BYOD policy to its implementation. Using a formalisation of the 5 policies written in SP4BYOD, we make comparisons between them, and explore the delegation relationships within them. We identify that whilst policy acknowledgement is a key part of all 5 policies, they are not managed by existing MDM tools.

1 Introduction

Employees bring their own devices to work. In the past employees might have had a dedicated company device; but today around 70% of companies have a BYOD scheme [27]. In some fields, 85% of staff use their personal devices to look up work-sensitive information [25]. Controlling employee's devices is a challenge for IT departments. Failure to manage devices can lead to employees accidentally leaking confidential information and the waste of company resources. Unfortunately the companies have limited control over the devices inside their networks if they do not own them.

One solution to controlling devices is requiring users agree to follow policies. They often take the form of a user agreement, written in natural language, which describes how devices should be used and configured. Various guides are available for companies wishing to implement a policy from governments, standards bodies, and organizations seeking to advise [23, 28, 9]. On top of user agreements, companies may also use Mobile Device Management (MDM) software which can help enforce policies. MDM software can configure a device's security settings, and add provide security APIs. But the use of MDM software does not guarantee compliance. One survey from a leading MDM vendor found over 50% of companies with their MDM software still had devices that did not comply with their policies [21]. Reasons for non-compliance included out-of-date policies and employees tampering with the MDM software.

BYOD policies are becoming more intricate. Prior work has looked at developing MDM software to enforce aspects of BYOD policies [11, 20, 4]. Implementing a policy is only part of the problem, however. BYOD policies are specified informally using natural language, and they contain more than just access control decisions. These policies describe trust relationships inside the company between the IT departments, users, and HR each who may be delegated to make decisions and provide rules. Policies contain rules that require employees to acknowledge risks, and regulations. An antivirus or MDM program may be used to *implement*

part of the policy. But it is the policy that *specifies* which software to use and when. There is no automatic way to check how the policy has been implemented and by what.

Companies lack visibility as to how they implement their policies. When considering what tools a company may use Morrow notes *"particularly with the BYOD trend IT professionals do not know if anti-virus software is installed or if it's current"* [22]. Even when devices can implement policies correctly, it is hard to configure devices that are not owned by the company [29]. Our work aims to address these problems directly: by using formal languages we can link the policy to the implementation.

To describe the policies we came up with SP4BYOD: a formal language for linking policies to the tools used to implement them and distributing decisions. Using a formalization of five BYOD policies written using SP4BYOD we identify different idioms and common delegation patterns present in BYOD policies. Our formalizations pick out the common concerns and trust relationships in these policies. We look at what decisions and trust relationships used in BYOD policies. We identify BYOD idioms that capture frequently seen decisions in BYOD policies. These give a guide for where future work implementing BYOD tools should focus their efforts to cover more aspects of policies.

SP4BYOD is not designed to replace existing static and dynamic analysis and enforcement tools. A company might use multiple different tools, app stores, and even contractual agreements with employees to enforce their policies. We aim to help clarify the meaning of ambiguous natural language policy documents, and provide a rigorous means for following them. A company can use any MDM tool, curated app store or user agreement to enforce their policy. SP4BYOD links the specification of the policy to its implementation, showing exactly how a policy is implemented and giving a rigorous means to enforce it.

We show how SP4BYOD can be used to encode these policies and describe precisely the different trust relationships. SP4BYOD is an instantiation of the SecPAL authorization language [7] for mobile device policies and implemented atop of AppPAL [13]. We have found, however, that SecPAL is a useful tool for describing other policies surrounding mobile ecosystems [14]. Our AppPAL implementation can be easily extended to support new types of policies. It also gives us access to tooling we have developed to check AppPAL policies for completeness and redundant statements. For this work additional tooling was developed to help visualise policies and describe their contents. This was helpful for making comparisons between policies and checking our formalisation for mistakes.

In summary, our work makes the following contributions:

- We present a formalisation of five different BYOD policies in SP4BYOD: a new instantiation of SecPAL for describing BYOD policies (section 2).
- Using our formalisation of the policies we make comparisons between the different policies. Unlike previous work which looks at individual policies [2], our work looks at policies across a variety of domains (section 4).
- We identify that delegation and acknowledgements are an important aspect of BYOD policies that current MDM software does not look at (section 6).

1.1 Related Work

Martinelli et al.'s work looks at creating a dynamic permissions manager, called UC-Droid. Their tool can alter what an app's Android permissions are at run time based on policies [20]. The tool allows companies to reconfigure their apps depending on whether the employee is at work, in a secret lab, or working out-of-hours. These kinds of policies are more configurable than the geofenced based policies some MDM tools provide. Other work has looked at enforcing different policies based on what roles an employee holds [11]. The work allowed a company to verify the devices within their network and what servers and services they could access. It also describes a mechanism for providing different users with different policies.

Armando et al. developed BYODroid as a tool for enforcing BYOD policies through a secure marketplace [3]. Their tool allows companies to distribute apps through a secure app store [4]. The store ensures apps meet policies through a combination of static analysis and app rewriting with dynamic enforcement. Their policies are low level, based on ConSpec [1], allowing checks based on Dalvik VM's state. Using their tool, they implemented parts of a NATO Communications and Information Agency policy relating to personal networks and data management [2]. Their work shows how the app-specific sections of a BYOD policy can be check and enforced using tools. They did not look at where the checks or policies come from, however.

An SP4BYOD policy might use BYODroid to ensure that parts of a policy are enforced (as well as other tools for other parts). Using SP4BYOD, we can distribute policies by sharing signed statements from different principals. We can delegate to other marketplaces to decide if an app meets different parts of policies. We can even create new stores by composing their policies and using multiple store's statements about the apps. Distributing checks like this is useful when using some static analysis tools which can take a long time to run (e.g. TaintDroid [12]).

Tools, such as Dr. Android and Mr. Hyde [18] and Aurasium [31], have suggested app wrapping (where an app is recompiled to use guarded APIs) as a possible way to enforce policies. App rewriting has the advantage that the device's underlying OS needn't be modified as the apps are changed at the source code level. However app wrapping alone without additional analysis is insufficient to enforce policies effectively [15].

Our approach taken with SP4BYOD is similar to work on safety cases. A safety case is an argument made to say a system is acceptably safe to be used in a given scenario. Industrial safety cases are often described in natural language, which can be ambiguous and unclear. Goal Structuring Notation (GSN) [**kelly˙goal˙2004**] is one approach to make the safety cases explicit. It is a graphical formal notation that lets engineers argue that a system is safe by linking safety goals to the arguments made for a system's safety. Similarly, work developing a formal language for specifying how medical staff should collaborate in a healthcare scenario [24] again helps clarify how roles are filled in a medical context on the basis of staff and different healthcare providers.

It is interesting to examine how leading [26] MDM tools such as IBM's MaaS360, or Blackberry Enterprise Services (BES), enforce *BYOD* policies. These tools support enforcing and checking compliance policies. They do not, however, use policy languages to specify policies; rather they provide a limited number of checkboxes that admins can tweak (an excerpt of a policy from MaaS360 is shown in Figure 1). These tools allow administrators to configure a device's settings and provision the devices with company apps. Some support app wrapping, which enables them to encrypt app data locally, use a VPN within the app, or prevent apps not being used when the device isn't compliant. But because the policies are inflexible and tightly coupled to the device's OS, intervention by an administrator is often required. Whilst MDM software is good at configuring devices, selecting which policies to apply is typically a manual process performed by an administrator. Removing blacklisted apps is a common feature, but the selection process of which apps to remove is manual.

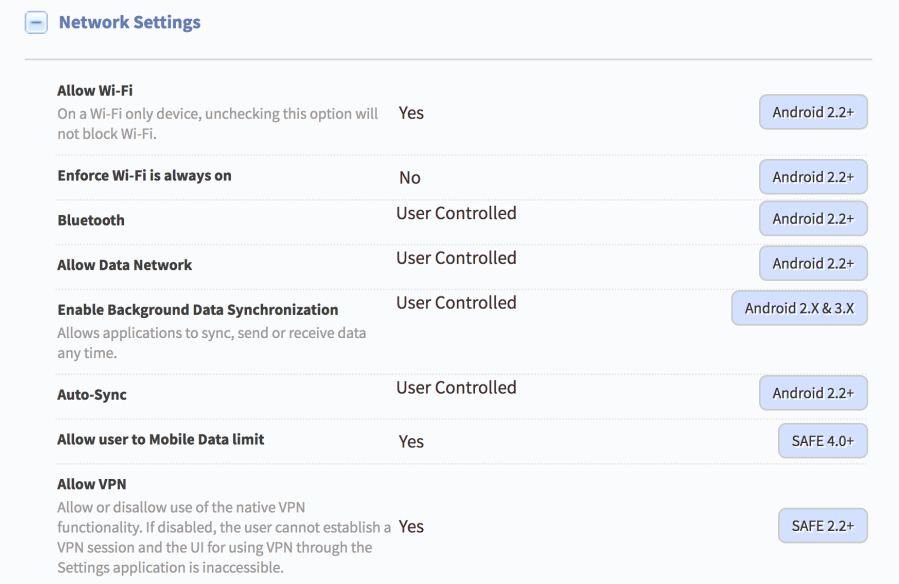


Fig. 1. Excerpt of a policy showing network settings from MaaS360.

2 Capturing BYOD Policies

As mobile devices have become more common in the workplace, BYOD policies have been written to help control them. Part of their policies are prescriptive: if you configure your device in this way, you will mitigate that threat. The policies contain more than just configuration, however. Consider this rule taken from

the *Security Policy for the use of handheld devices in corporate environments* by SANS [23].

> **SANS**: *Digital camera embedded on handheld devices* might *be disabled in restricted environments, according to* ⟨*COMPANY NAME*⟩ *risk analysis. In sensitive facilities, information can be stolen using pictures and possibly sent using MMS or E-mail services.*
>
> *In high-security facilities such as R&D labs or design manufacturers, camera MUST be disabled. Furthermore, MMS messages should be disabled as well, to prevent malicious users from sending proprietary pictures.*

A company could use an MDM program to enforce this. Some MDM tools can use geofencing to apply a policies in the area around a lab. Techniques like this would implement the recommendation within the rule, but the rule itself contains more than just configuration. It talks of *restricted environments* decided by *company risk analysis*. How is this communicated to the device? Does it access the list of restricted environments once from a server, are they fixed or can a device decide them for itself? Can it judge using a policy if a location is restricted? The rule also gives a security objective: *prevent malicious users from sending proprietary pictures*. The guidelines are given, however, for the case of a legitimate user using MMS or email. It may not be sufficient to stop a sufficiently motivated *malicious* user.

Our approach does not try to enforce the policy by checking the app's code for programming errors. Rather we act as a *"glue-layer"* between the high-level policy and the tools and trust relationships used to implement them. We capture the goals of the policy rules so that the delegations of trust, tools implementing the policy and their configuration are made explicit. This gives us greater clarity as to which tool is being trusted to implement what policy. It allows us to see who is being trusted to make which decisions, and use automatic-tools to uncover problematic aspects of the policy [14]. Continuing with the example above, we can encode this in SP4BYOD as:

```
'company' says 'risk-analyst' can-say
 Location:L isHighSecurityFacility.

'company' says Device:D mustDisableIn(Location, 'camera')
 if Location isHighSecurityFacility.

'company' says Device:D mustDisableIn(Location, 'mms')
 if Location isHighSecurityFacility.

'company' says User:U hasSatisfied('proprietary pictures policy')
 if U hasDevice(D),
    D mustDisableIn(Location, 'camera'),
    D mustDisableIn(Location, 'mms'),
    Location isHighSecurityFacility.
```

After checking the policy we generate a proof tree that shows how the policy was satisfied. These proof trees not only show how the policy was followed but

also provide an audit trail. In a company decisions may be delegated to different departments. Auditors can see what happened when things go wrong. They know who made what decision, and whether they made it through following policy rules or as a stated fact.

3 Instantiating SecPAL

SecPAL was developed as a distributed access control language [7]. It is designed to be have a clear readable syntax, and intuitive semantics. It is also designed to be extensible, which makes it ideal for extending to create new languages. All SecPAL statements are *said* by an explicit authority. The authority can say a fact (that something is described by a predicate), a delegation (that someone else *can-say* a fact), or a role assignment (that something *can-act-as* something else). This statement optionally contain conditional facts, and constraints that must be satisfied before the authority will say the statement.

To create SP4BYOD we instantiate SecPAL with four kinds of facts common in BYOD policies: *can, has, is* and *must*. Like other SecPAL-based instantiations [8, 6] we extend the syntax of facts to support these constructs.

Fact	Meaning
subject can*Action*	The subject is permitted to perform the action.
subject has*Action*	The subject has performed the action.
subject is*Type*	The subject is a member of the type.
subject must*Action*	The subject must perform the action.

Facts of the *must*-kind represent obligations, actions to complete if a particular scenario presents itself. For these facts, we add a rule to check we perform the obligation. This rule should be checked periodically to ensure compliance. Our implementation contains tooling to generate these rules automatically, by parsing the policy.

```
⟨speaker⟩ says ⟨subject⟩ hasSatisfiedObligation⟨Action⟩
  if ⟨subject⟩ must⟨Action⟩,
     ⟨subject⟩ has⟨Action⟩.
```

Facts using *is* predicates give types to variables. SP4BYOD inherits from SecPAL's (and Datalog's) safety condition that the body of a statements must reference all the variables in the head. This can lead to some *boilerplate* code in policies that may obscure their meaning. To simplify the policies, we add syntactic sugar for facts giving variables their type (*variable* is*Type*). Variables in the head of the statement of the form `Type:Variable` are replaced by the variable and a condition `Variable is`*Type* is added to the condition. The two statements shown below (taken from the SANS policy) are equivalent, however we feel the example on the right is easier to read.

```
'company' says Device
    canConnectToAP(X)
  if X isOwnedByCompany,
    Device isDevice,
    X isAP.
```

```
'company' says Device:D
    canConnectToAP(AP:X)
  if X isOwnedByCompany.
```

4 BYOD Policies

We examined 5 policies and encoded them into SP4BYOD looking for common idioms. We selected these policies as they came from a variety of domains.

- The first is the *Security Policy Template: Use of Handheld Devices in a Corporate Environment*, published by the SANS Institute [23]. This policy is a hypothetical policy published to help companies mitigate the threats to corporate assets caused by mobile devices. Companies are expected to modify the document to suit their needs. The policy is general; not specific to any particular industry, device, or country's legislation.
- The second is taken from the Healthcare Information Management System Society (HiMSS) [16]; a US non-profit company trying to improve healthcare through IT. The HiMSS policy is relatively short and contains concerns specific to healthcare scenarios. It is written as a contract the users agree to follow. In contrast, every other policy we looked at is written as an organisation imposing rules on users they should follow to ensure compliance. The policy is designed as a sample agreement for a system trying to manage personal mobile devices in a healthcare environment.
- The third is taken from a British hospital trust [19] and describes the BYOD scheme used in practice at the hospital.
- Finally, we looked at two simpler policies from The University of Edinburgh [30] and a company specialising in emergency sirens [10]. These policies are simpler, and shorter than the other policies we looked at comprised of much more general rules.

	SANS	HiMSS	NHS	Edinburgh	Sirens
Number of rules	33	15	56	20	25
SP4BYOD statements	71	21	58	10	39
Policy coverage	33 (100%)	14 (93%)	40 (71%)	10 (100%)[1]	22 (88%)
Rules using Acknowledgement	2	10	11	1	6
Rules using Delegation	23	5	33	2	13
Rules describing a restriction	18	3	8	1	5
Principal Speaker	company	user	nhs-trust	records-management department	

Table 1. Summary of the contents of each of the BYOD policies.

We summarise the policies in Table 1. Each policy contains a series of *rules*, which we implemented by one or more *SP4BYOD statements*. The *policy coverage* represents the number of rules that have an SP4BYOD description attached.

All five of the policies make use of acknowledgements. The use of an acknowledgements could be because enforcing that rule in a policy through technical means is undesirable. It could indicate policy authors care more that the subjects are aware of the rules than they do for rigorous enforcement. All but the HiMSS policy have rules that include locking down a device by disabling features. All but the Edinburgh policy have rules that look at what should happen if a user loses their device. The rest have rules that require employees inform someone when something happens. Common concerns, such as these, suggest where future MDM software should focus their efforts.

Only the NHS and SANS policies, the two most complex policies, describe when a device can install an app and what kinds of apps are installable. In both policies this expressed as a delegation to the appropriate groups to authorize an app. For example, in the SANS policy the IT-Department are responsible for deciding what apps can be installed. The NHS policy, however, is significantly more complicated. Apps have to be approved by three different groups (the IGC, the Employee's manager, and the relevant group for either clinical or business cases) before the Trust will say that an employee can install an app.

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 or other technologies that could benefit their patients/clients, this should be discussed with the clinical lead in the first instance and ratification should be sought via the Care and Clinical Policies Group. A clinical app should not be used if it has not been approved via this 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 or other technologies that could benefit their non-clinical work, ratification of the app must be sought via the Management of Information Group (MIG). An app should not be used if it has not been approved via this group. Following approval through Care and Clinical Policies and/or MIG, final approval will be required through Integrated Governance Committee. Use of paid apps must be agreed in advance with the device holder's line manager and there should be a demonstrable benefit.*

```
'nhs-trust' says App isUsable if App hasMet('clinical-use-case').
'nhs-trust' says App isUsable if App hasMet('business-use-case').
'nhs-trust' says 'cacpg' can-say App:A hasMet('clinical-use-case').
'nhs-trust' says 'mig' can-say App:A hasMet('business-use-case').
'nhs-trust' says App isInstallable
```

[1] The Edindiburgh policy contains a large nmuber of rules that whilst marked as rules are infact just descriptions of the document. All the policy rules that described restrictions or relationships were implemented in SP4BYOD.

```
  if App hasMet('final−app−approval'), App isUsable.
'nhs−trust' says 'igc' can-say App hasMet('final−app−approval').
'nhs−trust' says Device canInstall(App)
  if App isInstallable, App isApprovedFor(Device).
'nhs−trust' says Employee:Manager can-say
  App:A isApprovedFor(Device)
  if Manager isResponsibleFor(Device).
```

We might expect corporate policies to describe what apps can be installed in terms of the apps functionality. This does not appear to be the case, however. As part of selecting the apps, an IT department or group may choose to use advanced instrumentation and policies [4]. Alternatively, they may manually chose apps to form a curated app store as some MDM vendors allow. From the perspective of the policy, it is more important *who* makes the decision rather than *what* they chose, however.

5 Authorization Example

As a worked-example consider the NHS rules for finding approved apps (section 4). Suppose an employee, *Alice*, wished to get an app, *com.microsoft.office*, installed on their device. To do so, Alice would have to convince the device that:

```
'nhs−trust' says 'alices−phone' canInstall('com.microsoft.office').
```

Alice wishes to use the app for business so to satisfy the policy Alice must collect the following statements:

- `'nhs−trust' says 'com.microsoft.office' isInstallable.`
  For this, she needs a statement from the Management of Information Group that it has a business use-case. She also needs approval from the Integrated Governance Committee.
  1. `'mig' says 'com.microsoft.office' hasMet('business−use−case').`
  2. `'igc' says 'com.microsoft.office' hasMet('final−app−approval').`
- `'nhs−trust' says 'com.microsoft.office' isApprovedFor('alices−device').` To get this she needs a statement from the manager responsible for Alice's device (*Bob*) approving the app.
  3. `'bob' says 'com.microsoft.office' isApprovedFor('alices−device').`
  4. `'nhs−trust' says 'bob' isResponsibleFor('alices−device').`
- Additionally, she needs the following typing statements.
  5. `'nhs−trust' says 'com.microsoft.office' isApp.`
  6. `'nhs−trust' says 'bob' isEmployee.`

Alice obtains the statements by contacting each of the speakers. Each may either give her the statement she needs or may give her additional rules. For example, the MIG and IGC may be happy to state their statements (after a review). When checking if the app is an App in item 5, the NHS trust may be instead inclined to delegate further. They could reply that if the App is in the

Google Play store then they are convinced it is an app. Alice would then have to obtain additional statements if she wanted to prove this statement. As with SecPAL, all statements should have a signature from their speaker proving they said the statement. Alternatively, the speaker could refuse to give the statement, either because they do not believe it to be true, or they cannot give an answer. In this case, Alice would have to look for an alternative means to prove the statement or accept that they cannot install the app.

When the statements have been collected Alice can use a SecPAL inference tool (such as AppPAL[2]) to check the policy has been satisfied. The generated proof from the tool lets auditors review how the decision was made, and verify the decision-making process.

6 BYOD Idioms

When examining the policies, we noticed two particular idioms in many policies: acknowledgements and delegation. We describe both idioms in greater detail, and show how they can be implemented in SP4BYOD, below. MDM tools and research have focussed so far on implementing restrictions on apps and devices [17, 5, 20]. Implementing these controls is a vital aspect of BYOD policies and all 5 of the policies we looked at had rules that described restrictions (Table 1). Every policy also contained rules that required employees acknowledgements, however. Only the SANS policy (which is configuration focussed) contained more rules that required restrictions than acknowledgements. All the policies contained more rules featuring delegation relationships than functionality restrictions.

Delegation and Roles within Policies. Delegation is an important part of each of the policies. Each of the policies describes through rules how separate entities may be responsible for making some decisions. These rules can be a delegation to an employee's manager to authorize a decision (as in the NHS policy). It could be to technical staff to decide what apps are part of a standard install (as in the sirens and SANS policies).

SP4BYOD requires an explicit speaker for each statement. Speakers can delegate to others by making a statement about what they *can-say*. When translating the policies, the author of the policy is used as the primary speaker of the policy's rule (Table 1). For the HiMSS policy, where the user states what they will do rather than the company stating what they must, the user is the primary speaker. All the policies describe multiple entities that might make statements and delegate. With SP4BYOD policies any speaker can delegate a decision to another speaker (with restrictions on re-delegation). The delegation might be to a user to acknowledge a policy, or it might be to other groups in the company who are responsible for certain decisions.

In all the policies we looked at the majority of the decisions are made by three groups of speakers: the company, the IT-department, and the users or employees.

[2] https://github.com/apppal/libapppal

All the policies also delegate to a user (apart from HiMSS where the user is the primary speaker). The user is typically responsible for providing information, such as agreements to policies, reporting devices missing, and updating passcodes. In the Sirens, SANS and NHS policy each describe an IT-department who are delegated to make some decisions. The HiMSS policy describes an *xyz-health-system* who act similarly to an IT-department. These decisions are more varied and can overlap with the responsibilities of the company. In the NHS and SANS policies, the IT department is responsible for maintaining lists of activated devices. In the Sirens and SANS policies, the IT department maintains a list of what is installable on a device or not.

When a policy decision requires input from a third-party delegation is used. For example, an employee's manager has to authorise an app install. The SecPAL *can-say* statement is the basis for a delegation. We can ask the HR department to state who is someone's manager.

```
'company' says 'hr−department' can-say
 Employee:E hasManager(Employee:M).
```

If we wish to delegate to someone, we can add conditionals to the can-say statement that enforces any relationship between the delegating and delegated parties.

```
'company' says Manager can-say
 Employee canInstall(App:A)
 if Employee hasManager(Manager).
```

Acknowledgement. All the policies we looked at require their subjects to be aware and acknowledge certain rules or policies, and that the company may perform certain actions. For example, the NHS and HiMSS policies state that the organisation will wipe devices remotely to protect confidential information a user loses their device. Both policies also say that employees would lose personal information if they had it on the device and the company needed to erase it. The employee is required to be aware of this, and in the case of the HiMSS policy, agree to hold the company harmless for the loss.

Both the SANS and the siren-company policies use acknowledgements to link to other sets of rules that employees should follow. These policies are not further specified, and in the case of an acceptable use policy may be hard to enforce automatically. The SANS policy requires that all employees follow an email security, acceptable use, and an eCommerce-security policy. The Sirens policy expects an employee to use their devices ethically and abide by an acceptable use policy.

When there is a (usually separate) set of rules and concerns employees should be aware of acknowledgements are used. The company may not have wish to enforce these separate rules automatically, however. For instance, a company may have an ethical policy that says employees should not use devices for criminal purposes. The company is not interested in, or capable of, defining what is

criminal. They trust their employees to make the right decision and to be aware of the rules.

To implement these in SP4BYOD, a policy author creates two rules: the first stating their employees must have acknowledged the policy, the second delegating the acceptance of the policy to the employee themselves.

```
'company' says Employee:E mustAcknowledged('policy').
'company' says Employee:E can-say
  E hasAcknowledged('policy').
```

7 Conclusions

We have presented SP4BYOD: an instantiation of SecPAL for BYOD policies. Using an SP4BYOD formalization of 5 BYOD policies we have identified that whilst delegation and acknowledgement form a large part of written BYOD policies, existing BYOD tools ignore them. BYOD policies contain delegation and trust relationships that define who is responsible for making different decisions in a company. Sometimes that is administrators and technical staff deciding what to permit inside the company, and sometimes it is the user's themselves agreeing to follow a policy. Previous work has focussed on the technical staff's decisions and developing new ways to automate their decisions. Our work looks at the policies at a higher level tracking, managing and authorizing policies based on what people have said and what tools were run.

SP4BYOD improves upon existing MDM tools by allowing sophisticated delegation relations and by providing a declarative language for expressing policies. The language gives greater flexibility to policy authors and allows them to write policies that depend on other policies rather than predefined settings and groups. It lets us track what users have agreed to, what their policies are, how they are specified, and how they are satisfied.

Acknowledgements were used in all the policies, but were not a part of MDM tools. A purely speculative explanation for this might be that the people using the MDM software (the IT department) do not care about the acknowledgements, and that another department (HR perhaps) are responsible for tracking what corporate policies employees have agreed to and have their own methods for dealing with that. Future work will aim to further explore how these acknowledgements are used within a company and how to manage them in a practical manner.

Related systems, such as GSN described in subsection 1.1, use a graphical notation. Whilst SecPAL-based languages are designed to be readable, diagrams can help make authors write policies and auditors understand them. Future work will look at extending SecPAL's notation to create such diagrams and further aid readability.

References

[1] I. Aktug and K. Naliuka. "ConSpec A Formal Language for Policy Specification". In: *Electronic Notes in Theoretical Computer Science* (Feb. 2008).

[2] A. Armando et al. "Developing a NATO BYOD security policy". In: *International Conference on Military Communications and Information Systems*. May 2016.

[3] A. Armando, G. Costa, and A. Merlo. "Bring Your Own Device, Securely". In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. SAC '13. New York, NY, USA: ACM, 2013, pp. 1852–1858. ISBN: 978-1-4503-1656-9. DOI: 10.1145/2480362.2480707. URL: http://doi.acm.org/10.1145/2480362.2480707 (visited on 06/16/2016).

[4] A. Armando et al. "Enabling BYOD through secure meta-market". In: *ACM Conference on Security and Privacy in Wireless and Mobile Networks*. Aug. 2014.

[5] A. Armando et al. "Formal modeling and automatic enforcement of Bring Your Own Device policies". In: *International Journal of Information Security* (Aug. 2014).

[6] B. Aziz, A. Arenas, and M. Wilson. "SecPAL4DSA: A Policy Language for Specifying Data Sharing Agreements". In: *Secure and Trust Computing, Data Management and Applications*. Communications in Computer and Information Science 186. June 2011, pp. 29–36.

[7] M. Y. Becker, C. Fournet, and A. D. Gordon. "SecPAL: Design and semantics of a decentralized authorization language". In: *Journal of Computer Security* (Jan. 2010).

[8] M. Y. Becker, A. Malkis, and L. Bussard. *A Framework for Privacy Preferences and Data-Handling Policies*. Technical Report MSRTR2009128. Microsoft Research, 2009. (Visited on 10/10/2016).

[9] CESG. *BYOD Guidance: Good Technology*. Tech. rep. CESG, Mar. 2015.

[10] Code3PSE.org. *Sample BYOD Policy*. URL: http://www.code3pse.com/public/media/22845.pdf (visited on 10/14/2016).

[11] G. Costantino et al. "Towards enforcing on-the-fly policies in BYOD environments". In: *International Conference on Information Assurance and Security*. Dec. 2013.

[12] W. Enck et al. "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones". In: *ACM Transactions on Computer Systems* (June 2014).

[13] J. Hallett and D. Aspinall. "AppPAL for Android". In: *Engineering Secure Software and Systems*. Apr. 2016.

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

[15] H. Hao, V. Singh, and W. Du. "On the effectiveness of API-level access control using bytecode rewriting in Android". In: *ASIA CCS* (2013).

[16] Healthcare Information and Management Systems Society. "Mobile Security Toolkit: Sample Mobile Device User Agreement". In: *Healthcare Information and Management Systems Society* (2012).

[17] *IBM MaaS360 - Enterprise Mobility Management (EMM)*. URL: http://www-03.ibm.com/security/mobile/maas360.html (visited on 10/12/2016).

[18] J. Jeon et al. "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.

[19] G. Kennington et al. *Mobiles Devices Policy*. Tech. rep. Torbay, Southern Devon Health, and Care NHS Trust, Mar. 2014.

[20] F. Martinelli, P. Mori, and A. Saracino. "Enhancing Android Permission Through Usage Control: A BYOD Use-case". In: *Symposium on Applied Computing*. 2016.

[21] MobileIron Security Labs. *Q4 Mobile Security and Risk Review*. Tech. rep. MobileIron Security Labs, Dec. 2015.

[22] B. Morrow. "BYOD security challenges: control and protect your most sensitive data". In: *Network Security* 2012 (Dec. 2012).

[23] Nicholas R. C. Guerin. *Security Policy for the use of handheld devices in corporate environments*. Tech. rep. SANS, May 2008.

[24] P. Papapanagiotou and J. D. Fleuriot. "Formal verification of collaboration patterns in healthcare". In: *Behaviour & Information Technology* 33.12 (Dec. 2014).

[25] R. K. Patel et al. "A UK perspective on smartphone use amongst doctors within the surgical profession". In: *Annals of Medicine and Surgery* (June 2015).

[26] Rob Smith et al. *Magic Quadrant for Enterprise Mobility Management Suites*. Tech. rep. G00279887. Gartrer, June 2016. URL: https://www.gartner.com/doc/reprints?id=1-390IMNG&ct=160608&st=sb (visited on 11/23/2016).

[27] H. Schulze. *BYOD & Mobile Security 2016 Spotlight Report*. Tech. rep. LinkedIn Information Security, 2016.

[28] M. Souppaya and K. Scarfone. "Guidelines for Managing and Securing Mobile Devices in the Enterprise: NIST Special Publication 800-124 Revision 1 (Draft)". In: *National Institute of Standards and Technology* ().

[29] B. Tokuyoshi. "The security implications of BYOD". In: *Network Security* 2013 (Apr. 2013).

[30] D. Williamson, A. Grzybowski, and S. Graham. *Bring Your Own Device Policy*. Policy 15. University of Edinburgh, Feb. 2015. URL: http://www.ed.ac.uk/files/imports/fileManager/BYODPolicy.pdf (visited on 10/14/2016).

[31] R. Xu, H. Sadi, and R. Anderson. "Aurasium: Practical Policy Enforcement for Android Applications". In: 2012.