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, this is 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. Today around 70% of companies have a BYOD scheme [1]. In some fields, 85% of staff use their personal devices to look up work-sensitive information [2]. Controlling employee's devices is a challenge for IT departments. Failure to manage devices can lead to employees accidentally leaking confidential information. Unfortunately companies have limited control over the devices inside their networks if they do not own them.

One solution to controlling devices is requiring users follow BYOD policies. A BYOD policy takes 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 [3, 4, 5]. 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 security APIs, helping enforce some aspects of the policies. But the use of MDM software does not guarantee compliance. One survey from a leading MDM vendor found over 50% of companies using their MDM software still had devices that did not comply with the policy [6]. Reasons for non-compliance included out-of-date MDM configurations that hadn't been updated, and employees tampering with the MDM software.

BYOD policies are becoming more intricate. Prior work has looked at developing MDM software to enforce some aspects [7, 8, 9]. The MDM encoding of 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. They describe trust relationships inside the company between IT

* Work supported by EPSRC App Guarden grant (EP/K032666/1) and the Alan Turing Institute.

departments, users, and HR, who each may be delegated to provide rules and make decisions. 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 well 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"* [10]. Even when devices can implement policies correctly, it is hard to configure devices that are not owned by the company [11]. Our work aims to address these problems directly: by using formal languages we can link the policy to the implementation.

To describe the BYOD policies we present SP4BYOD: a formal language for linking policies to the tools used to implement them, and distributing decisions to relevant parties. 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 are 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 MDM 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 tools, app stores, or 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 policies and describe precisely the different trust relationships. SP4BYOD is an instantiation of the SecPAL authorization language [12] for mobile device policies and implemented atop of AppPAL [13], which adds mobile device specific predicates to SecPAL, for example, capturing app permissions. SecPAL is also a useful tool for describing other policies surrounding mobile ecosystems [14]. Our SP4BYOD implementation can be easily extended to support new types of policies. It also gives us access to tooling we have developed to check SP4BYOD 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 [15], 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 [8]. 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 [7]. 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 [16]. Their tool allows companies to distribute apps through a secure app store [9]. 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 [17], 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 [15]. 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 [18]).

Tools, such as Dr. Android and Mr. Hyde [19] and Aurasium [20], 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 [21].

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) [22] 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 [23] 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 [24] 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.

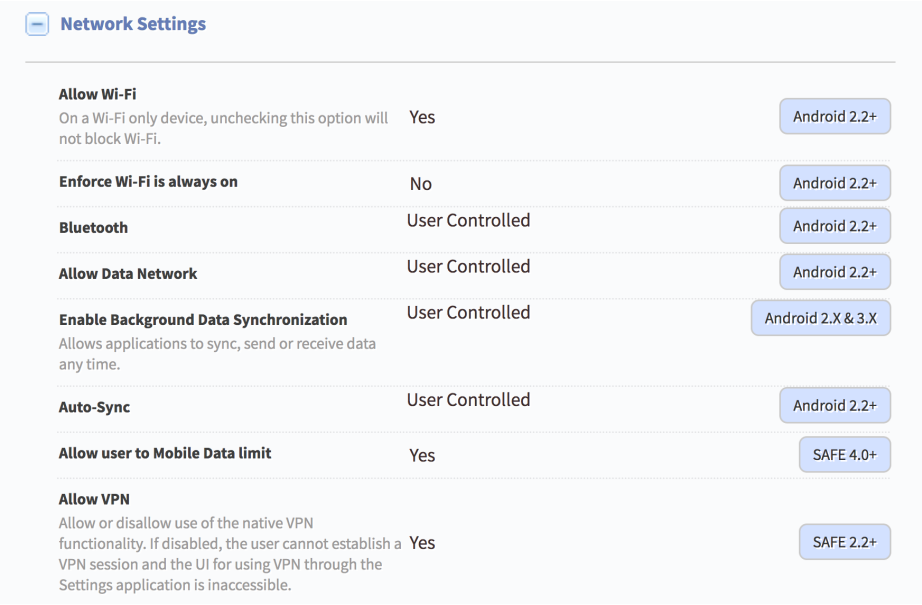

Fig. 1. Excerpt of a policy showing network settings from MaaS360.

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 [3].

> **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 [12]. 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 [25, 26] 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 [3]. 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) [27]; 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 [28] and describes the BYOD scheme used in practice at the hospital.
- Finally, we looked at two simpler policies from The University of Edinburgh [29] and a company specialising in emergency sirens [30]. These policies are simpler, and shorter than the other policies we looked at comprised of much more general rules.

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

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

	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.

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
  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 [9]. 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 [31, 32, 8]. 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

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

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. 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 wish to enforce these separate rules automatically, however. For instance, a company may have an ethics 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] H. Schulze. *BYOD & Mobile Security 2016 Spotlight Report*. Tech. rep. LinkedIn Information Security, 2016. URL: {http://static.tenable.com/whitepapers/byod-and-mobile-security-report-2016.pdf}.

[2] 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).

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

[4] 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* ().

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

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

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

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

[9] A. Armando et al. "Enabling BYOD through secure meta-market". In: *ACM conference on Security and Privacy in wireless & mobile networks*. Aug. 2014.

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

[11] B. Tokuyoshi. "The security implications of BYOD". In: *Network Security* 2013.4 (Apr. 2013), 12–13.

[12] 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).

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

[16] 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.

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

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

[19] 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.

[20] R. Xu, H. Sadi, and R. Anderson. "Aurasium: Practical Policy Enforcement for Android Applications". In: *Usenix Security Symposium*. 2012.

[21] H. Hao, V. Singh, and W. Du. "On the effectiveness of API-level access control using bytecode rewriting in Android". In: *ACM Asia Conference on Computer and Communications Security* (2013).

[22] Tim Kelly and Rob Weaver. "The goal structuring notationa safety argument notation". In: *Proceedings of the dependable systems and networks workshop on assurance cases* (2004). (Visited on 12/19/2016).

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

[24] 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`.

[25] M. Y. Becker, A. Malkis, and L. Bussard. *A Framework for Privacy Preferences and Data-Handling Policies*. Technical Report MSRTR2009128. Microsoft Research, 2009.

[26] 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.

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

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

[29] 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).

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

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

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