Obligations to Enforce Prohibitions: On the Adequacy of Security Policies

Wolter Pieters
Delft University of Technology
& University of Twente
The Netherlands
w.pieters@tudelft.nl

Julian Padget
University of Bath
United Kingdom
jap@cs.bath.ac.uk

Francien Dechesne
Delft University of Technology
The Netherlands
f.dechesne@tudelft.nl

Virginia Dignum
Delft University of Technology
The Netherlands
m.v.dignum@tudelft.nl

Huib Aldewereld
Delft University of Technology
The Netherlands
h.m.aldewereld@tudelft.nl

ABSTRACT

Security policies in organisations typically take the form of obligations for the employees. However, it is often unclear what the purpose of such obligations is, and how these can be integrated in the operational processes of the organisation. This can result in policies that may be either too strong or too weak, leading to unnecessary productivity loss, or the possibility of becoming victim to attacks that exploit the weaknesses, respectively. In this paper, we propose a framework in which the security obligations of employees are linked directly to prohibitions that prevent external agents (attackers) from reaching their goals. We use graph-based and logic-based approaches to formalise and reason about such policies, and show how the framework can be used to verify correctness of the associated refinements. The framework can assist organisations in aligning security policies with their threat model.

1. INTRODUCTION

Security policies in organisations typically take the form of obligations for the employees. However, it is often unclear what the purpose of such obligations is, and how these can be integrated in the operational processes of the organisation. This can result in policies that may be either too strong or too weak, leading to unnecessary productivity loss, or the possibility of becoming victim to attacks that exploit the weaknesses, respectively.

The primary goal of any security policy is to specify means for facing a given environment of threats. When organisations wish to protect their information assets against malicious attacks, the first step is stating what should be protected against what. For example, an organisation may wish to prevent outsiders from gaining access to sales data. In order to ensure that such constraints hold, organisations then take concrete measures that actually reduce access possibilities, such as locks on doors, access control on IT systems, and rules for employee behaviour. These security measures again determine the possibility or impossibility of gaining access, but at a more detailed level. The question then becomes how the threat model and security measures can be aligned. In particular, this holds for policies imposed on employees.

A first attempt to formalise the notion of security policy alignment, using a formalisation in first order predicate logic, is presented in [17]. The authors discuss consistency and completeness of policies expressed at different levels in an organisation. For example, the organisational policy that sales data should not leave the organisation may be refined into policies on passwords, door locks, and employee behaviour. In [17], the authors mainly focus on preventive controls (such as locks and passwords), but their framework does not include the possibility of expressing obligation alongside permission and prohibition. In order to deal with policies in the form of obligations for employees, we need to adapt the approach.

In this paper, we propose a framework in which the security obligations of employees are linked directly to prohibitions that prevent external agents (attackers) from reaching their goals. We show why obligations are an essential addition to the framework when trying to model complex organisations. We view the organisation as a system, composed of agents which can perform some actions on some objects, to be regulated by a security policy. A security policy on such a system aims at defining what actions the agents are permitted, obliged or forbidden to perform [4]. We use a formal representation, which has the advantage of being able to define precisely how to reason about a policy, and therefore to develop analytical tools to study the consequences of a given policy. We formalise the problem of verifying completeness, and show how this can be addressed in both graph-based and logic-based analysis. This provides answers to questions like "do these policies address the threats" and "what if I remove this policy", thereby determining whether policies are too strong or too weak.

As a running example, we discuss stealing a laptop from an office [8]. An attacker may try to obtain a key to open the door, or enter the room when it is unlocked. Stealing the laptop is not possible when the owner is in the office, even when the door is unlocked. Locks are important in this case, but they do not work without the obligation to lock the room when leaving the office. Note that, although this case addresses physical features of information security for illustration purposes, the framework is applicable to digital controls and associated obligations as well.

In section 2, we discuss related approaches, and the differences with the present framework. In section 3, we outline the basic concepts for representation and analysis, which we formalise in section 4. In section 5, we outline a graph-based approach to policy anal-

ysis, and in section 6 a complementary logic-based approach. In section 7, we evaluate the results of both approaches, and in section 8, we discuss applications of the framework and draw conclusions.

2. RELATED WORK

Our work builds on the notion of security policy alignment. Alignment of security policies can be discussed for policies in different domains, or at different levels of abstraction. In the first case, one may for example wish to align policies for digital access and physical access [15]. In the second case, one may wish to investigate whether the digital and physical access policies match the policy that sales data should stay within the organisation. The former can be called horizontal alignment, and the latter vertical alignment [7]. In the case of vertical alignment, it may be the case that only the policies at a higher level of abstraction are known, and that the policies at a lower level need to be designed. This constitutes the activity of refinement: defining lower-level policies that should correctly implement a higher-level policy.

Security policy refinement was already identified in [1], but not formalised. Consistency and completeness of policies is discussed in [21], including the notion of refinement. The question of alignment was taken to the socio-technical domain by Dimkov [7], who aimed at integrating policies on digital assets, buildings, and employee behaviour. This approach was formalised in [17]. These approaches focused on permission and prohibition in relation to automated methods for attack path discovery ("attack navigators"), but did not include the notion of obligation. Obligation and its relation to responsibility is discussed in [5, 20].

In this paper, we primarily study the translation of prohibitions into obligations, and the delegation of those obligations to agents in the system. From a more philosophical perspective, such problems have been discussed from the framework of actor-network theory, describing socio-technical systems with a symmetrical view on people and things. In this context, Latour speaks of "programs" and "antiprograms", similar to our notions of agents under control and agents outside control. Latour's example is hotel keys, with the security problem of guests not returning them (antiprogram). One can then delegate the responsibility for seeing to it that guests return their keys (program) to, for example, signs or bulky key rings [12, 13]. The refinement of security policies can be thought of as an approach to include safety and security values in the design of systems. Therefore, the proposed methods can be used in approaches such as value-sensitive design [9].

3. BASIC CONCEPTS

In this section, we define the basic concepts that form the foundations of our approach.

3.1 System and control

First of all, we make a distinction between the area ("system") that is under control of the agent defining the policies, and the area that is outside her control. When defining and implementing policies, the focus is always on the area that is under one's control. The behaviour of attackers cannot be influenced, and the only way to limit the behaviour of attackers is by indirect means, i.e. implementing policies in the form of access control mechanisms, door locks, and rules on the behaviour of employees. In this paper, the focus is on how obligations for employees can enforce prohibitions for attackers.

It must be noted that the attacker may actually be an "insider" [18], i.e. someone from within the organisation. This does not invalidate the assumption of control, as long as the boundaries are properly defined. For example, one may assume that at most k employees collude in an attack, and that the others follow the imposed policies.

3.2 Obligations as refinements of prohibitions

Whereas physical and digital security policies are mostly expressed as permission and prohibition, many *organisational* security policies take the form of obligations. Prohibitions expressed in high-level policies may be translated into obligations in lower-level policies. Obligations, like prohibitions, can only be attributed to those who are under your control. Attackers cannot be assumed to follow any of their obligations, so it is not relevant to assume they have any. So, prohibitions on the attacker translate into obligations for the employees, and the obligations somehow enforce the high-level prohibition.

Obligations are targeted at keeping the system in a "safe" state (the obligation to *see to it that* (STIT) nothing bad happens). In particular, states in which an agent outside the control of the organisation can initiate an action that would violate a security policy should be avoided. For the running example, we have at least three possibilities to prevent theft of a laptop from a room (safe states), assuming the attacker does not have the key:

- employee in room, laptop in room;
- nobody in room, laptop in room, room locked;
- no laptop in room.

The assumptions are that an attacker cannot steal a laptop from a room while somebody is there, that the attacker cannot break the lock, and that the attacker cannot steal the laptop when it is not in the room. Obviously, this also assumes that there is no other way to get into the room except through the door with the lock. The question in the running example is how to design obligations for the employees that would be a suitable refinement of the prohibition of attackers to take away the laptop. Or, a more modest question, whether a particular set of obligations would be a valid refinement of said prohibition.

In a generalised form, our question is thus how to refine a high-level policy expressed as prohibition on agents outside control, into lower-level policies, in the form of obligations, for people inside control. Expressing such policies and refinements requires not only a suitable logic, but also a suitable ontology (that provides an inventory of what agents inside and outside the organisation can and cannot do).

3.3 Obligations and agency

In this paper, we assume that any agent can be assigned an obligation, including non-human agents. For the sake of space, we ignore here any philosophical questions concerning agency. In this context, the door/room/laptop example may yield the following obligations:

- The door is obliged not to let anyone in without a key when locked;
- The employee is obliged to lock the door whenever leaving the room;
- The employee is obliged not to let anyone she does not know take something from the room while present.

4. FORMALISATION

4.1 The formal framework

Figure 1 gives an overview of the relationships between the different components in which we frame the security policy situation. Solid arrows are translation steps, dotted arrows are verification steps. We use X to refer to a member of the set of *eXternal* agents $\mathcal{X}$, and E for agents under control (*employees*, in set $\mathcal{E}$). A prohibition on Xs (upper left) is translated into a high level obligation (STIT) on Es (upper right), which in turn is refined into policies for individual agents E (lower right). Agents conforming with these policies enable or disable certain behaviours for agents X (lower right), which may or may not completely realise the prohibition on X (upper left).

Formalising the components, i.e. making structure explicit in formal languages, allows us to make these relationships explicit and more precise at the same time. Ultimately, the goal of formalisation is to: (i) verify consistency and completeness between layers, and (ii) to assist in (automatically) generating refinements of the prohibitions into policies (obligations).

At the higher level (also: higher abstraction level), security requirements impose limitations on an attacker X: it is forbidden for X to ϕ – where $\phi(X)$ stands for some undesirable state of affairs brought about by X. A policy holding for those subject to it (i.e. the Es, not the Xs), dictates that the Es make sure that $\phi(X)$ does not come about: the Es should *see to it that* $\neg\phi(X)$. For now, we take the arrow on the top level to be a generic, yet informal rule: from a security requirement forbidding an attacker X to ϕ, follows the policy that those under control see to it that X does not ϕ. This approach suggests the application of a deontic logic of agency, *STIT*-logic, which has '*seeing to it that*' as primitive component.

4.2 Logics for obligations and prohibitions

In the top level of Fig. 1, we use logical formalism to express prohibitions, responsibilities and obligation. Deontic logic started out with actions as primitives, but when it was discovered that a logic of permission and obligation fitted with modal Kripke semantics, actions were replaced by states of affairs, like our $\phi(X)$ [22, 23]. The STIT-operator ([3]) constitutes a responsibility relation between agents and states of affairs (conditions). So, the actions that lead to a state of affairs (like $\phi(X)$) are left implicit, and no distinction is made between different ways of bringing about the same result.

Logics of agency like STIT need temporal order to be interpreted, in the form of a branching time semantics. This is necessary to model the agent's ability to choose (a certain action to bring about a certain state of affairs). A choice by an agent at a certain moment is a subset of futures; $STIT_E\psi$ basically corresponds to the possibility of E choosing only futures in which ψ holds [19]. In this paper we focus only on the policies on the lower level, so the semantics of the formulas on the top level remains informal. In future work, we aim to formalise the higher level and connect it to the lower level through formal semantics for the top-level formulas.

4.3 Formalising the problem

In this section, we formalise the concepts outlined above. First of all, we distinguish two disjoint sets of agents, where the organisation is in control of its employees ($E \in \mathcal{E}$), but not in control of externals $X \in \mathcal{X}$.

1. $\mathcal{E} \bigcap \mathcal{X} = \emptyset$
2. $\mathcal{E} \bigcup \mathcal{X} = \mathcal{A}$ (where $\mathcal{A}$ is the set of all agents)
3. $control(E)$ for each $E \in \mathcal{E}$
4. $\neg control(X)$ for each $X \in \mathcal{X}$

We postulate here that a prohibition $F_X\ \phi(X)$ implies an obligation and responsibility $O_{\mathcal{E}}\ STIT_{\mathcal{E}}(\neg\phi(X))$ (translation from prohibition for X to $\phi(X)$ into obligation on E's to maintain not $\phi(X)$). Note that this is a postulate here: as the formulas have no formal semantics yet, we cannot formally validate this implication. We aim to do so in future work.

From the top level, we now need refinement of the general obligation and responsibility of the employees into policies for each agent. This may involve refining ϕ as a conjunction of multiple states that would enable the outsider/attacker X to achieve $\phi(X)$.

4.4 Analysis workflow

Given the obligation of the agents E to see to it that agents X do not achieve a certain state of affairs, this high-level obligation needs to be refined into more detailed obligations, distributed over all agents in $\mathcal{E}$. Two types of analysis are possible using the models: a completeness analysis of proposed detailed obligations, and an analysis providing suggestions on which actions should be prevented by the obligations. The analysis requires the following input:

1. The property of the world to be prevented, typically an agent X having access to a certain asset;
2. A model of the world/system describing which actions by the agents are possible;
3. The proposed obligations on the agents in $\mathcal{E}$ to prevent agents X from achieving the property under 1.

Obligations may take the form of limitations on certain possible actions of the agents E, either being obliged to abstain from the action (don't leave the room), being obliged to verify certain conditions before taking the action (leave the room only when there are no valuables), or doing the action only in combination with another action (lock the room when leaving).

For the completeness analysis, it needs to be clear which actions are prevented by the defined obligations. For the analysis providing suggestions for obligations, this link also needs to be available in the other direction: given a certain action, can it be prevented by a certain obligation? Actions can only be prevented by obligations if an agent E is involved. Actions taken by an agent X on its own can never be prevented by obligations on agents in $\mathcal{E}$. The analysis aims at proposing a minimal set of obligations that would be complete with respect to the high-level obligation to prevent the attacker goal. This amounts to blocking all paths in the attack tree by obligations on the agents in $\mathcal{E}$. In this paper, we assume that this is the only way to prevent attacks, and we do not take technical countermeasures into account (changing what is possible in the system).

4.5 Finding counterexamples

In the running example, the high-level policy states that an attacker should not get access to the laptop. Assuming that the employee residing in the office will always lock the door when leaving with the laptop still in the room, one would assume that the system would always be in a state where (i) the laptop is not in the room, (ii) the employee is in the room, or (iii) the room is locked, and that therefore the laptop is safe. However, depending on the system model / ontology and the actions that it allows, there is a counterexample. Consider the situation where the attacker enters the room while the employee is still there with the laptop, and then the employee leaves without the laptop and locks the door. The attacker will then have access to the laptop. Whether he can actually steal it will depend on whether he can unlock the door from the inside without the key.

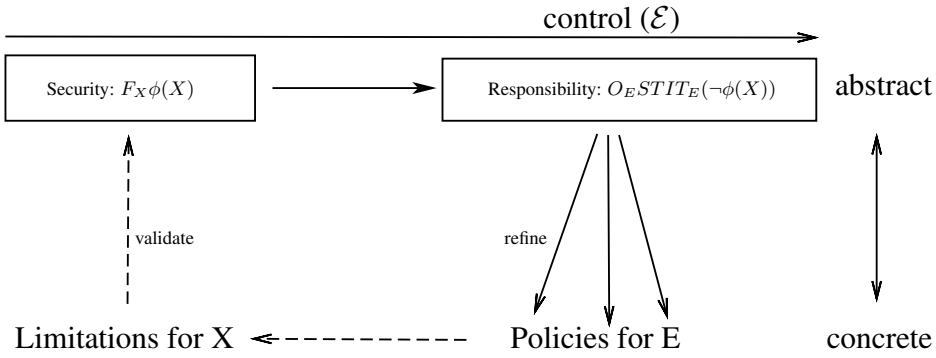

Figure 1: Overview of the refinement of prohibitions into obligations. Solid arrows represent design steps, dotted arrows represent verification steps.

5. THE EXAMPLE IN ANKH

The ANKH system model [16] represents security in socio-technical systems by means of hypergraphs, where nodes are entities and hyperedges represent access relations. When somebody is in a room, she will be a member of the hyperedge representing the room. Entities that are members of more than one hyperedge (guardians) will have policies stating on which conditions they allow entities to move between hyperedges. For example, the door is a member of the hallway and the room, and will have policies stating how somebody can enter (or exit), e.g. by possessing the right key.

For the purpose of modelling our running example, four extensions are required to the original ANKH model:

1. We need actions that change policies: locking the door means changing the policy of the door (it will now require a key). In particular, actions can change the state of an entity, and policies are associated with states;
2. We need asymmetric policies: entering a room may require different credentials (e.g. a key) than leaving;
3. We need a notion of simultaneous action: the employee is required to lock the door *at the same time* as leaving the room;
4. We need policies that require the *absence* of something: the attacker cannot take the laptop while the employee is there.

Here, we represent access policies as tuples (g, e, C, S, T), meaning that guardian g will give entity e access to target T if e has access to all elements in C (credentials), and the elements in S (surveillers) have no access to e. Additionally, there are meta-policies for changing the policies. The door has two policies for granting access to the room, and someone with the key can make the door switch policies. Thus, we represent such a meta-policy as (P, C), with P a set of policies and C a set of sets of credentials, where each set of credentials in C is sufficient to make the door switch to a different *active policy*.

In the starting state, the room contains the door, the laptop, and the employee, the employee's possessions include the key, and the hallway contains the door and the attacker (Fig. 2). The door is open at this stage, and having access to the hall is sufficient to gain access to the room (no key is required). The active policy of the door can be changed by (un)locking. We assume that the door never requires a key to exit, or rather that anyone who has access to the room can change the policy of the door (lock or unlock from the inside), without needing the key. Formally, $C = \{\{key\}, \{room\}\}$.

The analysis of the example in ANKH forces us to make all policies explicit. For example, we need to state that the employee can pick up or leave any object, and that anyone can exit the room (without the key). We assume that the hallway is large enough for the attacker to take an asset, even with the employee present in the hallway. Based on this analysis, we can identify all possible sequences of actions, and associated states of the system. We can then provide an undesirable property of a state as the target of an attack analysis, in this case the attacker having access to the laptop. Tracing back the target to possible preconditions, the analysis then builds an attack tree [14] (Figure 3).

6. THE EXAMPLE IN INSTAL

Whereas ANKH uses a graph-based approach to policy analysis, logic-based approaches can serve a similar purpose, using an exhaustive analysis of the reachable states, given an initial state and an ontology of actions available to the actors. Inst*AL* is an action language, following in the tradition of $\mathcal{A}$ [10] and the Event Calculus [11]. Inst*AL* is implemented in Answer Set Programming [2] and was originally developed to support institutional modelling [6]. Informally, Inst*AL* is a simple set-theoretic model for event-based systems, such that an event brings about a change in a state that models the situation of interest. The apparent compatibility of the underlying formal models of ANKH and Inst*AL* suggested the idea of capturing the movements of entities subject to policy as events that modify a representation of the states arising from the possible enactments of policies. Answer set semantics conceptually grounds the input program over all the values that the variables can take, effectively constructing a proof tree, in which the paths from root to leaf constitute the answer sets for the model under consideration. Consequently, it is possible to explore all possible sequences of events entailed by some starting state, reflecting the intuition expressed in section 4.2. Of course, as described, this leads to exponential blow-up, however, many event orderings are either meaningless or just not of interest, so constraints expressing properties over paths that are not wanted can be used to prune the proof tree, reducing the number of answer sets significantly.

The way we have used Inst*AL* here is as a means to specify how significant events change the state of the model *vis-à-vis* the security of certain entities. Hence, by specifying a range of initial conditions, it is possible to derive traces that capture the effects of all the event orderings considered interesting and thus whether undesired states are reachable or not. In this sense, modelling the example in Inst*AL* is similar to the ANKH analysis, but using logic rather than graphs. The basic concepts are expressed as predicates

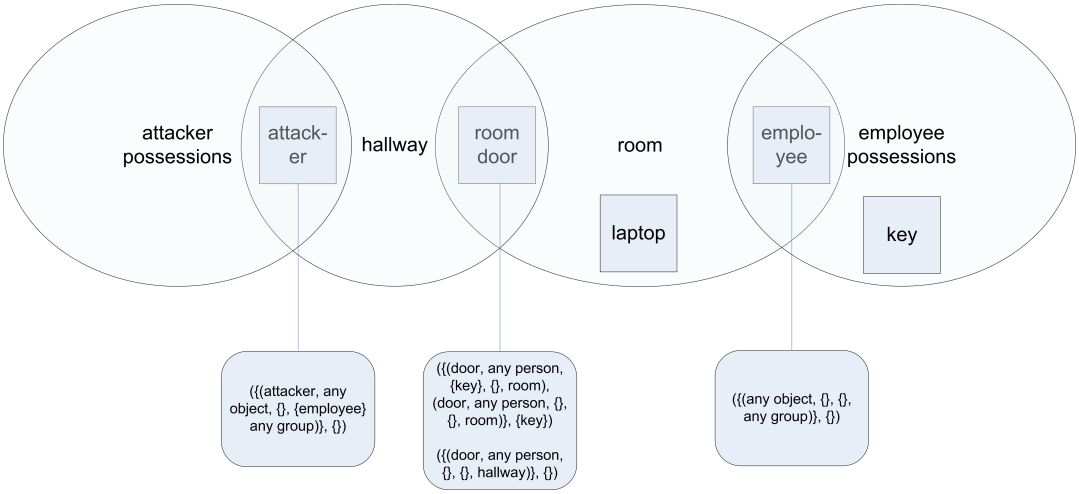

Figure 2: The initial state of the example as an ANKH hypergraph.

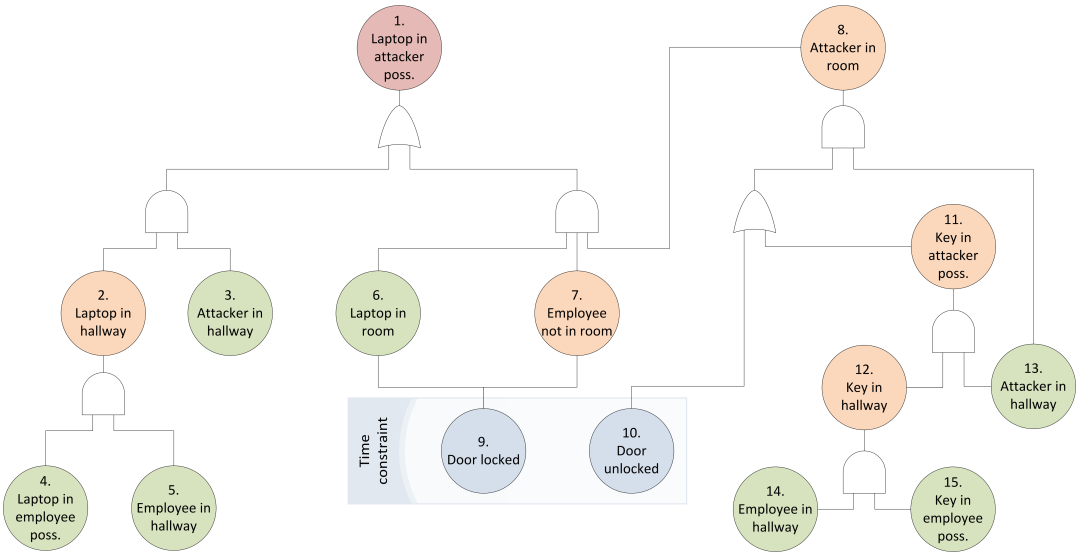

Figure 3: The attack tree of the example as developed by the ANKH analysis. AND-nodes have a flat basis (all children need to be satisfied), and OR-nodes are curved (only one child needs to be satisfied). Nodes 9 and 10 represent a time / sequence constraint, as they cannot both be realised at the same time.

that can be derived from properties of the state. In particular, we regard the protection mechanisms to have `failed` when the attacker gains possession of the laptop:

```
failed when holds(A,O),
   attacker(A),
   laptop(O)
```

`failed` is an institutional fact, whose presence in the institutional state indicates that the resource – in this case the laptop – is now held by a non employee. Throughout the program fragments, we use `A` to denote an Agent (either an employee or an attacker), `O` an Object (either a key or a laptop) and `L` a Location (either an office or a hallway). What matters is how `holds(A,O)` is achieved; that is what was the sequence of events that brought about the state containing `failed`? The fact that an attacker holds the resource is brought about by the attacker `take`ing the resource when it is in a `vulnerable` situation:

```
take(A,O) initiates holds(A,O)
   if alocation(A,L), olocation(O,L),
      vulnerable(O), attacker(A);
take(A,O) initiates holds(A,O)
   if alocation(A,L), olocation(O,L),
      not holds(A,O), employee(A);
```

This rule applies to any resource, not just a laptop and hence also covers the case of the key to the office being vulnerable. A resource is `vulnerable` when an attacker can `take` it. This is reflected by defining `vulnerable` as the disjunction of several situations:

```
vulnerable(O) when situation1(O);
vulnerable(O) when situation2(O);
vulnerable(O) when situation3(O);
vulnerable(O) when situation4(O);
```

Situation 1 arises when an employee leaves a resource unattended in an unlocked office. Hence, the attacker can enter the office and `take` the resource.

```
situation1(O) when
   olocation(O,L),
   not employeeIn(L),
   alocation(A,L),
   unlocked(L)
```

Situation 2 allows an attacker to be alone in a locked office with the resource, so the attacker can `take` the resource.

```
situation2(O) when
   olocation(O,L),
   not employeeIn(L),
   alocation(A,L),
   locked(L)
```

It is clear that the flaw in the security policy here is in locking the office with an attacker inside, but it is a situation that is not specified by the initial form of the policy resulting in a `failed` state in the analysis and a trace that identifies how the situation arose. Consequently, the designer may specify an additional obligation for the employee to ensure this situation is prevented (see section 7)

In **situation 3**, the laptop has been left in the hallway. This makes it vulnerable because anyone may `take` it.

```
situation3(O) when
   olocation(O,L), laptop(O), hallway(L)
```

In **situation 4**, the key has been left in the hallway. If the attacker can `take` the key, then they can enter the (locked) office and `take` the laptop.

```
situation4(O) when
   olocation(O,L), key(O), hallway(L)
```

The above characterise several situations of concern. What matters from a security analysis point of view is how any one of these situations may be arrived at from an initially secure situation; that is to say, what is the trace of events that takes the system situation from secure to insecure? Consequently, by analysing those traces, the policy can be refined to oblige employees to take or not to take certain actions in order to avoid bringing about a failed state.

We start from the initial facts:

```
initially
  olocation(laptop1,office1),
  alocation(agent1,office1),
  alocation(agent2,hallway1),
  hallway(hallway1),
  office(office1),
  unlocked(office1),
  holds(agent1,key1),
  key(key1),
  laptop(laptop1),
  employee(agent1),
  attacker(agent2)
```

In this situation, the laptop is in the office, with an employee who also has the key to the office, while the attacker is in the corridor. By running the answer set solver with the rules shown in figure 4 and the above initial conditions, we can discover all the traces that have `failed` in the final state. Without any constraints the number of answer sets increases by 2^4 (for this problem, because the maximum number of variables in any right hand side of a rule is happens to be 4) with each increment in the trace length, so a trace of length n has $2^{4(n-1)}$ answer sets. However, as noted earlier, not all sequences of events make sense and so we define several constraints to discard the corresponding answer sets (or rather, to ensure they are not constructed in the first place).

The events that trigger the initiation or termination of institutional facts in the model have no semantic import for the answer set solver: they are just names of terms and it constructs answers sets based on all the possible orderings of those names. However,

```
enter(A,L) initiates unlocked(L),alocation(A,L)
   if office(L),holds(A,K),key(K);
enter(A,L) initiates alocation(A,L)
   if unlocked(L),office(L);
enter(A,L1) terminates locked(L1),alocation(A,L2)
   if locked(L1),office(L1),
      holds(A,K),key(K),hallway(L2);
enter(A,L1) terminates alocation(A,L2)
   if unlocked(L1),office(L1),hallway(L2);

exit(A,L1) initiates alocation(A,L2)
   if unlocked(L1),office(L1),hallway(L2);
exit(A,L1) initiates alocation(A,L2),locked(L1)
   if office(L1),hallway(L2),holds(A,K),key(K);
exit(A,L) terminates alocation(A,L)
   if unlocked(L),office(L),alocation(A,L);
exit(A,L) terminates alocation(A,L),unlocked(L)
   if unlocked(L),office(L),
      holds(A,K),key(K),alocation(A,L);

take(A,O) initiates holds(A,O)
   if alocation(A,L), olocation(O,L),
      vulnerable(O), attacker(A);
take(A,O) initiates holds(A,O)
   if alocation(A,L), olocation(O,L),
      not holds(A,O), employee(A);
take(A,O) terminates olocation(O,L)
   if alocation(A,L), olocation(O,L);

leave(A,O) initiates olocation(O,L)
    if holds(A,O), alocation(A,L);
leave(A,O) terminates holds(A,O)
    if holds(A,O);
```

Figure 4: Behavioural rules

it makes no sense, for example, to enter the same place in successive time instants[1], or to enter a location if already there. Similar common sense constraints apply to the action of exiting a location and taking and leaving objects. Here we consider traces of length 6, since this happens to be the number of events required to illustrate the scenario of the attacker taking the key that the employee leaves in the hallway and entering the office containing the laptop (see Figure 5: a + prefix identifies a fluent that is initiated in a given instant, conversely a - prefix denotes termination, while **bold** identifies a non-inertial fluent when it is added to the state). Shorter and longer scenarios are equally possible: it is a matter of both the total number of different kinds of events that are specified and lengths of the sequences of interest to the designer. Traces of length 6 give rise to 2^{20} answer sets, but the common-sense filters reduce this to 30.

7. ANALYSIS RESULTS

In the preceding sections, we have shown how both graph-based and logic-based approaches can be used to analyse the completeness of the obligations with respect to enforcing the prohibition on the external agents. Although the graph-based and logic-based approaches have different notations, both have representations of policies and actions, which together provide the possibility to find problematic sequences of actions. In ANKH, this result is represented in an attack tree (Figure 3), in Inst*AL* in the form of traces (Figure 5). Examination of, respectively, the attack tree and the traces that lead to a failed state reveals three policy oversights: the one that we already discussed and two additional ones:

[1]Time necessarily advances, but there is no notion of real time, rather events are associated with an identified instant and instants are ordered.

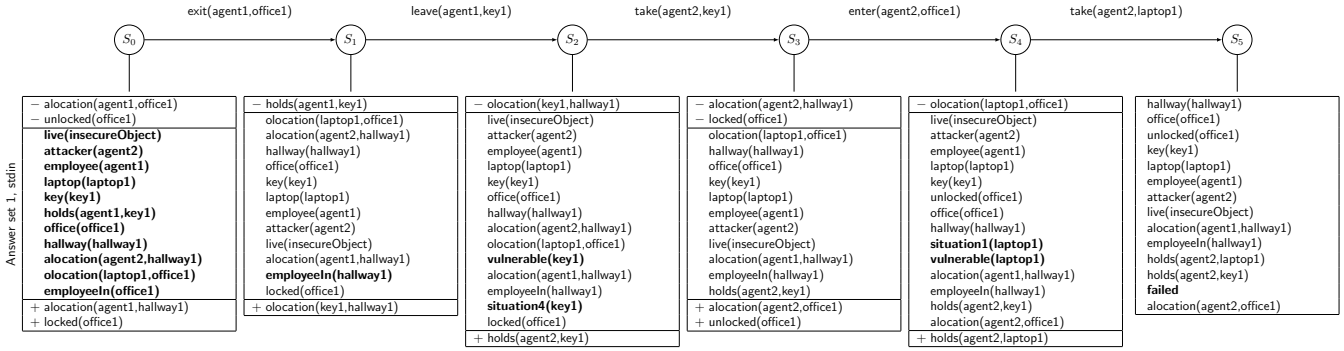

Figure 5: Trace shows employee leaving key in hallway, attacker taking key, entering office and taking laptop.

- attacker enters office, employee leaves and locks door; attacker takes laptop (assuming he can unlock from the inside);
- employee leaves laptop in hallway; attacker takes laptop;
- employee leaves key in hallway; attacker takes key, enters room while employee is away, and takes laptop.

Thus, in order to make the STIT predicate true, and enforce the prohibition on the attackers, three additional obligations need to be assigned to the employee:

- Don't leave the room when someone else is there. In Inst*AL* , this can be expressed by adding a condition to the exit rule:

```
1 exit(A,O) terminates alocation(A,O),unlocked(O)
2    if unlocked(O),office(O),
3       holds(A,K),key(K),alocation(A,O),
4       not attackerIn(O);
5
6 attackerIn(L) when alocation(X,L), attacker(X);
```

  Taking the same initial conditions as before and the same common-sense filter rules, the number of traces with failed in the final state now reduces to 3.

- Don't leave the key in the hallway. In Inst*AL* , this can be expressed by adding a condition to the leave rule:

```
1 leave(A,O) initiates olocation(O,L)
2     if holds(A,O),
3        alocation(A,L), not hallway(L);
4 leave(A,O) terminates holds(A,O)
5     if holds(A,O),
6        alocation(A,L), not hallway(L);
```

  Adding this rule results in no answer sets with failed in the final state (in traces of length 6).

- Don't leave the laptop in the hallway. This is covered by the above rules because they apply to any object.
- The usefulness of this process in uncovering incompleteness of the original policy is evidenced by the revelation of another failure case: when the employee leaves the key in the office and so cannot lock the office when she leaves as she is obliged to do. Consequently, the attacker can enter the office and take the laptop. This scenario can be resolved by obliging the employee not to leave the key *anywhere*.

In the ANKH analysis, these additional obligations are added to the policies of the hypergraph entities. This prevents the corresponding actions (graph transformations) from occurring, and thereby removes the corresponding branches from the attack tree (Figure 3).

For the first additional obligation, node 7 gets an additional child "attacker not in room". This adds an additional time constraint, requiring the employee to leave before the attacker enters. The latter is only possible if the attacker acquires the key, as the employee is obliged to lock the door. Adding the second additional obligation would remove nodes 2 and 12, removing the possibility that either the key or the laptop is left outside. Similarly to the Inst*AL* analysis, adding both obligations thereby blocks all paths.

8. CONCLUSIONS AND DISCUSSION

In this paper, we have outlined a framework for systematic description and analysis of obligation policies on employees that result from the desire to prohibit actions of agents that are outside of the control of the organisation. This prohibition is translated into obligation on the agents that are indeed under control, and we can verify the completeness of such a refinement with tool support. We have shown that both graph-based and logic-based approaches can be used to execute the analysis. The framework can be applied in different settings. As we have shown above, it can be employed to check the completeness of obligation policies put on employees, with respect to the goal of preventing undesired states. Similarly, it could be used to verify whether a policy set would still be complete when removing one entry, thereby identifying superfluous policies.

Based on this verification approach, a method could be developed that would assist policy developers in step-wise design of complete policies. As we have shown above, the gap analysis provided by the framework can be used to adjust or add policies, and then the analysis can be re-run to check whether the problems have been solved, or whether new problems might appear. In this way, policy designers can use the framework to develop better policies. Also, if there is discussion on certain policy, for example when employees complain about the burden it brings, or when it is systematically ignored, the framework can be used to show what could go wrong if the policy would not be there. Assuming that certain employees (in agent set $\mathcal{E}$) would not comply with their obligations, one could analyse the possible traces to undesirable states that would be enabled. This would for example be relevant in the analysis of insider threat. The analysis could be part of a general pro-active approach to identify what could happen if any of the employees would not comply, or it could be part of an investigation when a concrete suspicion about insider threat would exist.

In future work, we focus on including support for step-wise refinement of the prohibitions into obligations for employees. We also aim at investigating the use of more advanced logics with semantics for obligation and responsibility, in order to explore the translation arrow on the top level of Fig. 1. Can we characterize the

kind of prohibitions on X that are formally translatable into obligations on E? This could give insight in limitations of what can be solved with security policies.

Acknowledgments

This research has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreements number SEC-261696 (SESAME) and ICT-318003 (TREsPASS). This publication reflects only the authors' views and the Union is not liable for any use that may be made of the information contained herein. The research also received funding from the Dutch Next Generation Infrastructures Foundation (project 09.08.KID).

9. REFERENCES

[1] M. Abrams and D. Bailey. Abstraction and refinement of layered security policy. In M. A. et al., editor, *Information Security: An Integrated Collection of Essays*, pages 126–136. IEEE Computer Society Press, 1995.

[2] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.

[3] N. Belnap, M. Perloff, and M. Xu. *Facing the Future: Agents and Choices in our Indeterminist World.* Oxford University Press, 2001.

[4] L. Cholvy and F. Cuppens. Analyzing consistency of security policies. In *IEEE Symposium on Security and Privacy*, pages 103–112, 1997.

[5] L. Cholvy, F. Cuppens, and C. Saurel. Towards a logical formalization of responsibility. In *Proc. of the 6th int. conf. on Artificial intelligence and law*, ICAIL '97, pages 233–242, New York, NY, USA, 1997. ACM.

[6] O. Cliffe, M. De Vos, and J. Padget. Answer set programming for representing and reasoning about virtual institutions. In *CLIMA VII*, volume 4371 of *LNCS*, pages 60–79. Springer, 2006.

[7] T. Dimkov. *Alignment of Organizational Security Policies – Theory and Practice*. PhD thesis, University of Twente, Enschede, February 2012.

[8] T. Dimkov, W. Pieters, and P. Hartel. Effectiveness of physical, social and digital mechanisms against laptop theft in open organizations. In *GreenCom, IEEE/ACM Int'l Conference on Cyber, Physical and Social Computing(CPSCom)*, pages 727 –732, dec. 2010.

[9] B. Friedman. Value-sensitive design. *Interactions*, 3(6):16–23, 1996.

[10] M. Gelfond and V. Lifschitz. Action languages. *Electron. Trans. Artif. Intell.*, 2:193–210, 1998.

[11] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, 1986.

[12] B. Latour. Where are the missing masses? the sociology of a few mundane artifacts. In *Shaping technology/building society: Studies in sociotechnical change*, pages 225–258. Cambridge MA: MIT Press, 1992.

[13] B. Latour, P. Mauguin, and G. Teil. A note on socio-technical graphs. *Social Studies of Science*, 22(1):33–57, 1992.

[14] S. Mauw and M. Oostdijk. Foundations of attack trees. In *Information Security and Cryptology - ICISC 2005*, volume 3935 of *LNCS*, pages 186–198. Springer, 2006.

[15] V. Nunes Leal Franqueira and P. A. T. van Eck. Towards alignment of architectural domains in security policy specifications. In J. M. P. et al., editor, *Proc. 8th Int. Symp. System and Information Security*. Fundacao Casimiro Montenegro Filho - CTA/ITA, 2006.

[16] W. Pieters. Representing humans in system security models: An actor-network approach. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 2(1):75–92, 2011.

[17] W. Pieters, T. Dimkov, and D. Pavlovic. Security policy alignment: A formal approach. *Systems Journal, IEEE*, 7(2):275–287, 2013.

[18] C. W. Probst, R. R. Hansen, and F. Nielson. Where can an insider attack? In *Formal Aspects in Security and Trust*, volume 4691 of *LNCS*, pages 127–142. Springer, 2007.

[19] K. Segerberg, J.-J. Meyer, and M. Kracht. The logic of action. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2012 edition, 2012.

[20] M. Sloman. Policy driven management for distributed systems. *Journal of network and Systems Management*, 2(4):333–360, 1994.

[21] M. Sloman and E. Lupu. Security and management policy specification. *Network, IEEE*, 16(2):10 –19, mar/apr 2002.

[22] G. von Wright. *Norm and Action: A Logical Enquiry*. Routledge & Keegan, 1963.

[23] J. Wolenski. Deontic logic and possible worlds semantics: A historical sketch. *Studia Logica*, 49(2):273–282, 1990.