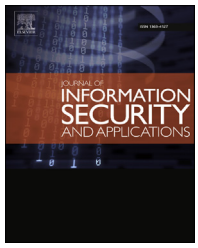
Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/jisa



Effectiveness of qualitative and quantitative security obligations

W. Pieters [a,b,*], J. Padget [c], F. Dechesne [d,e], V. Dignum [a], H. Aldewereld [a]

[a] ICT – Department of Technology, Policy and Management, Delft University of Technology, The Netherlands
[b] Services, Cyber Security and Safety – Electrical Engineering, Mathematics and Computer Science, University of Twente, Enschede, The Netherlands
[c] Department of Computer Science, University of Bath, United Kingdom
[d] Energy & Industry – Department of Technology, Policy and Management, Delft University of Technology, The Netherlands
[e] eLaw@Leiden, Leiden Law School, Leiden University, The Netherlands

ARTICLE INFO

Article history:
Available online 23 August 2014

Keywords:
Graphs
Logics
Obligations
Prohibitions
Refinement
Security policies

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 logic-based and graph-based approaches to formalise and reason about such policies, and show how the framework can be used to verify correctness of the associated refinements. Finally, we extend the graph-based model with quantitative policies and associated quantitative analysis, based on the time an adversary needs for an attack. The framework can assist organisations in aligning security policies with their threat model.

1. Introduction

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 Pieters et al. (2013a). The authors discuss consistency and completeness of policies expressed at

* *Corresponding author*. ICT – Department of Technology, Policy and Management, P.O. Box 5015, 2600 GA Delft, The Netherlands.
E-mail addresses: w.pieters@tudelft.nl (W. Pieters), jap@cs.bath.ac.uk (J. Padget), f.dechesne@tudelft.nl (F. Dechesne), m.v.dignum@tudelft.nl (V. Dignum), h.m.aldewereld@tudelft.nl (H. Aldewereld).
http://dx.doi.org/10.1016/j.jisa.2014.07.003
2214-2126/

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 Pieters et al. (2013a), 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 formalise the problem of verifying completeness, and show how this can be addressed in both logic-based and graph-based analyses. This provides answers to questions like "do these obligations address the threats" and "what if I remove this obligation", determining whether policies are too strong or too weak.

As a running example, we discuss stealing a laptop from an office (Dimkov et al., 2010). 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. For simplicity, we assume only one office and one key (to that office). There are numerous possible extensions, e.g. several laptops, different types of property that can be stolen, existence of security cameras, etc., but it is not the purpose of this paper to solve all such extensions; rather the scenario serves to illustrate the approach, and therefore is only as simple as necessary for this purpose. In the laptop theft example, technical measures (locks) are only effective in combination with 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.

Compared to the paper presented at the SIN'13 conference (Pieters et al., 2013b), we provide additional models and analyses for quantitative policies, in relation to tool support for attack-defense trees in ADTool (Kordy et al., 2013). In particular, we show how policies can be annotated with time metrics, in order to overcome the limitations of binary (yes/no, possible/impossible) policies. For this purpose, we have also extended the related work section. This paper does *not* address the question whether quantitative policies are in general better or more effective than qualitative policies. Rather, we extend the options for analysing the effectiveness of policies from qualitative to quantitative ones.

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 and the formal framework based on those. In Section 4, we outline a logic-based approach to policy analysis and evaluate the results of this approach. In Section 5, we outline a complementary graph-based approach. We extend this model for quantitative policies in Section 6, and present the associated analyses in Section 7. In Section 8, we discuss the possibilities for further extension of the (quantitative) models, and in Section 9, we investigate applications of the framework and draw conclusions.

2. Related work

Our work builds on the notion of security policy alignment. Such 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 (Nunes Leal Franqueira and van Eck, 2006). 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 (Dimkov, 2012). 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 Abrams and Bailey (1995), but not formalised. Consistency and completeness of policies is discussed in Sloman and Lupu (2002), including the notion of refinement. The question of alignment was taken to the socio-technical domain by Dimkov, 2012, who aimed at integrating policies on digital assets, buildings, and employee behaviour. This approach was formalised in Pieters et al. (2013a). 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 Cholvy et al. (1997), Sloman (1994), Feltus et al. (2010).

The relation between security policies and attack scenarios was discussed in Pieters et al. (2013a); Kammüller and Probst, 2013). In particular, these papers interpret attack scenarios as those scenarios that violate policies, or, vice versa, policies as constraints on acceptable behaviours. Attack trees (Mauw and Oostdijk, 2006) are tree structures describing sets of attack scenarios. Leaf nodes are atomic attack steps, and these can be combined with AND- and OR-nodes to show their relation to the attacker's goal in the root of the tree. In (Pieters et al. 2013a), it was observed that attack trees and security policies are equivalent in the sense that they represent sets of behaviours (and implicitly their (in)acceptability). Attack trees can be augmented with defenses, yielding attack-defense trees (Kordy et al., 2012). Attack trees and attack-defense trees can be annotated with quantitative properties, such as probability of success and time required. For building trees and running analyses tool support is available (Kordy et al., 2013).

Quantitative security policies were first introduced in Degano et al. (2011), focussing on probability of violation in a stochastic process calculus. We apply the notion here for security policies in a socio-technical context, in which agents may have obligations to enforce quantitative constraints on other agents.

3. Formal framework

3.1. Model structure

We now turn to the formal model of prohibition and obligation policies. First of all, we make a distinction between the area ("system") that is under control of the organisation defining the policies, and the area that is outside its control. A security policy on such a system aims at defining what actions the agents are permitted, obliged or forbidden to perform (Cholvy and Cuppens, 1997). 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 they will not comply with prohibitions or obligations. 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.

Fig. 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 adopt the naming conventions of deontic logic (Meyer et al., 1993), where O stands for Obligation, F for Prohibition (forbidden) and P for Permission. We use *X* to refer to a member of the set of *eXternal* agents not under control $\mathscr{X}$, and *E* for an agent under control (*Employees*, in set $\mathscr{E}$), with $\mathscr{E} \cap \mathscr{X} = \emptyset$ and $\mathscr{E} \cup \mathscr{X} = \mathscr{A}$ (where $\mathscr{A}$ is the set of all agents). A prohibition on Xs (upper left) is translated into a high level obligation on Es (upper right), which in turn is refined into obligations for individual agents *E* (lower right). Agents conforming with these policies enable or disable certain behaviours for agents *X* (lower left), which may or may not completely realise the prohibition on *X* (upper left). Ultimately, the goal of formalisation is to: (i) verify consistency and completeness between layers, and (ii) 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 happen: 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.

We postulate here that a prohibition $F_x\phi(X)$ implies an obligation and responsibility $O_E STIT_E(\neg\phi(X))$ (the obligation on Es to maintain $\neg\phi(X)$). 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 $\phi(X)$. 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.

3.2. 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 $\mathscr{E}$. Two types of analysis are possible in our approach: a completeness analysis of the proposed detailed obligations, and an analysis providing suggestions of which actions should be prevented by the obligations. The analysis requires the following input:

1. The state 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 $\mathscr{E}$ to prevent agents *X* from achieving the property under 1.

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 the running example, we assume the following initial obligations for employees:

- 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.

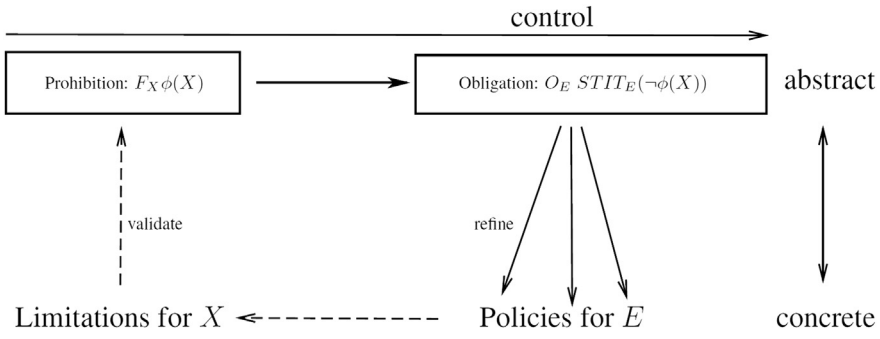


Fig. 1 – Overview of the refinement of prohibitions into obligations. Solid arrows represent design steps, dotted arrows represent verification steps.

In general, obligations may take the form of limitations on 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 $\mathscr{E}$. 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's goal.[1] This amounts to blocking all attack paths by obligations on the agents in $\mathscr{E}$ (which could be both technical and human agents).

4. The example in InstAL

4.1. The InstAL language

As the formal framework is based on obligations and prohibitions, logic-based reasoning seems a natural choice for supporting the analysis. Logic-based approaches can serve the purpose of assisting in the completeness analysis, using an exhaustive analysis of the reachable states, given an initial state and an ontology of actions available to the actors. InstAL is an action language, following in the tradition of $\mathscr{A}$ (Gelfond and Lifschitz, 1998) and the Event Calculus (Kowalski and Sergot, 1986). InstAL is implemented in Answer Set Programming (Baral, 2003) and was originally developed to support institutional modelling (Cliffe et al., 2006). Informally, InstAL 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. This suggests 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 3.1. Of course, as described, this leads to exponential blow-up[2]; 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. In this exploration of the approach, these constraints are defined by hand (as we explain later, see Section 4.2); there is no semantic information associated with the events as far as an Answer Set Solver is concerned: the only association with the scenario is in the naming chosen by the modeller. Furthermore, since such constraints are always going to reflect properties of the domain being modelled, fully automatic derivation would imply a substantial reasoning process. However, the task could be semi-automated through the use of the institutional query language (InstQL) (Hopton et al., 2009).

4.2. Modelling the example

The way we have used InstAL for our running example 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. The basic concepts are expressed as predicates 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). For simplicity, we assume only one office and one key (to that office). What matters is how `holds(A,O)` is achieved: 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 (we use the term *situation* to refer to a description of a state (or states) of the institution that satisfies (satisfy) some condition):

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

[1] In this paper, we present an approach to the construction of this set of obligations, but proof of minimality is outside the scope of the paper.

[2] Since the purpose of Answer Set Solvers is to enable the exhaustive exploration of state spaces, complexity immediately falls into the NP-complete class, as with any other model-checking-like process.

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)
```

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), not employeeIn(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), not employeeIn(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 Fig. 2 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), as we now explain.

```
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);
```

Fig. 2 – Behavioural rules.

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, it makes no sense, for example, to enter a location if already there (see Fig. 3, line 2). Similar common sense constraints apply to the action of exiting a location and taking (see Fig. 3, lines 4 and 6) 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 Fig. 4). 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.

4.3. *Analysis results*

The representations of policies and actions provide the possibility to find problematic sequences of actions. In InstAL,

```
   % A cannot enter O if already in O
:- observed(enter(A,O),I),holdsat(alocation(A,O),I).
   %  A cannot take O if A already holds O
:- observed(take(A,O),I),holdsat(holds(A,O),I).
   % A1 can't take O if A2 holds O
:- observed(take(A1,O),I), holdsat(holds(A2,O),I), A1!=A2.
```

Fig. 3 – Examples of some "common-sense" constraints.

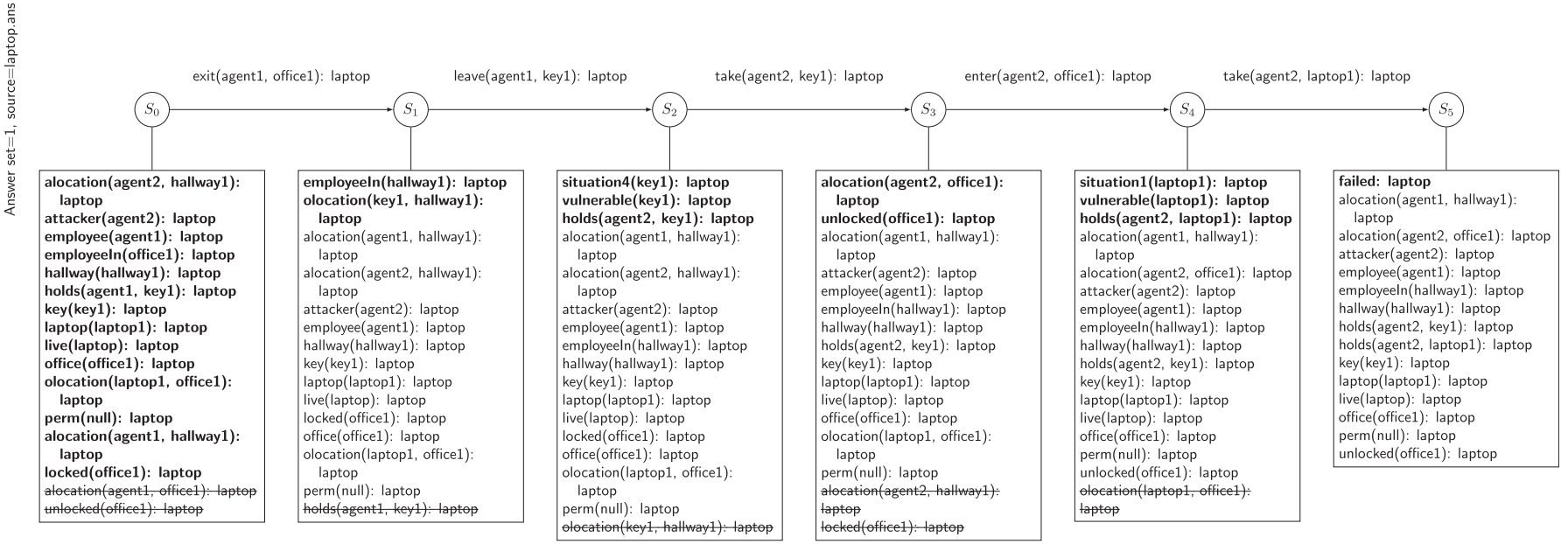


Fig. 4 – Trace shows employee leaving key in hallway, attacker taking key, entering office and taking laptop. Bold identifies a fluent that is initiated in a given instant, while strikethrough denotes termination.

this result is represented in the form of traces (Fig. 4). Examination of the traces that lead to a `failed` state reveals three policy oversights:

- 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.

The first one may be the most surprising: the attacker enters the room while the employee is still there with the laptop, and the employee leaves without the laptop and locks the door. Assuming the attacker can unlock the door from the inside without the key, he can now steal the 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 InstAL, this can be expressed by adding a condition to the exit rule:

```
exit(A,O) terminates alocation(A,O),unlocked(O)
   if unlocked(O),office(O),
      holds(A,K),key(K),alocation(A,O),
      not attackerIn(O);

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 InstAL, this can be expressed by adding a condition to the leave rule:

```
leave(A,O) initiates olocation(O,L)
    if holds(A,O),
       alocation(A,L), not hallway(L);
leave(A,O) terminates holds(A,O)
    if holds(A,O),
       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 analysis so far is useful and also illustrates the benefit of expressing goals as states to be achieved or avoided so that the corresponding model can help uncover additional obligations or refine existing ones. It does however lack a quantitative dimension: actions appear to be instantaneous and can follow one another immediately without taking account of the duration that an action physically requires. Thus while the qualitative approach enables high level exploration of sequences of states, it currently lacks the necessary degree of fidelity if obligations are to have temporal attributes, such as "employees should see to it that an attacker cannot obtain possession of a laptop in less than 3 min".

Given the exhaustive model generation approach provided by the Answer Set solver, it is important to avoid the use of unbounded integer computations. Thus, it might be indicated that `holds(A,O)` is true after `3` time steps:

```
take(A,O) initiates after(3,holds(A,O))
   if alocation(A,L), olocation(O,L),
      vulnerable(O), attacker(A);
```

but combining temporal annotations, as in:

```
take(A,O) initiates after(T1+T2,holds(A,O))
   if alocation(A,L,T1), olocation(O,L,T2),
      vulnerable(O), attacker(A);
```

is unsafe. A further, although not insurmountable complication, is the desire to annotate the activities with real numbers to denote, say, minutes and fractions of a minute, but such numerical values are not directly supported in ASP. Hence, we defer the integration of quantitative obligation attributes into the logic model for further work and (in the next section) focus on the specification of the scenario as an attack tree and show how the quantitative annotations should be interpreted.

5. The example in ANKH

5.1. *The ANKH model*

As an alternative for quantitative policies, we consider the graph-based ANKH system model (Pieters, 2011). We choose this particular framework, because (a) ANKH can generate attack trees from system models, and (b) the quantitative analysis of attack trees is a well-established field, including tool support.

For qualitative policies, modelling the example in ANKH is similar to the InstAL analysis, but using graphs rather than logic. The ANKH system model 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, called 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.

The present case uses the extensions to the original ANKH model as described in Pieters et al. (2013b). 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 (locked and unlocked), 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

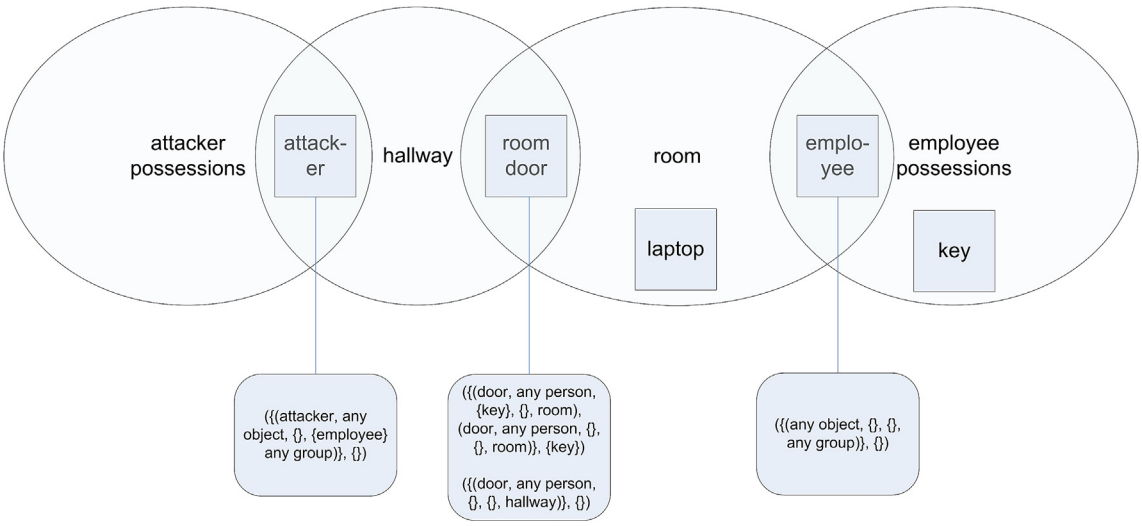


Fig. 5 – The initial state of the example as an ANKH hypergraph.

(Fig. 5). 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, $C = \{\{key\},\{room\}\}$.

5.2. *Attack trees, goals, states, and actions*

Based on the analysis of the policies, 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 (Pieters, 2011). Tracing back from the target to possible preconditions, the analysis then builds an attack tree (Mauw and Oostdijk, 2006). The worst-case complexity of this analysis is $O(N^6)$ with standard model checking tools, and $O(N^4)$ with dedicated tools. In practice, experiments with a similar modelling framework give a measured complexity between $O(N^{1.7})$ and $O(N^{3.3})$, depending on the properties of the model (Pieters et al., 2013a).

In our attack tree, the children of a node represent the preconditions (subgoals) needed to enable a certain action, as well as the action itself. Thus, the parent node describes the *postcondition* of an action for which the children are the *preconditions* and the *action*. These can be connected as an AND-relation. In addition, when multiple precondition-action combinations can lead to the same postconditions, this can be represented as an OR-node. The attack tree for the example, represented in ADTool (Kordy et al., 2013), is shown in Fig. 6.

Note that this type of analysis forces one to consider different types of nodes in attack trees, namely conditions and actions. By contrast, traditional attack trees typically consider actions only, at different levels of abstraction. When generating an attack tree from ANKH, leaf nodes are still actions, but parent nodes would rather represent states, state attributes or goals. The actions are then ways to satisfy these attributes, rather than being more abstract actions, composed of the actions represented by the children. However, as only leaf nodes are in the end part of the semantics of attack trees, this is not so much of a problem. Originally, attack trees would just represent sequences or multisets of actions (Mauw and Oostdijk, 2006), which only include the labels of the leaf nodes. In this sense, it is perfectly consistent to label intermediate nodes with goals or attributes rather than actions.

5.3. *Adding obligations*

Based on the attack tree, we can identify the same issues as with InstAL in Section 4.3. These correspond to the attack paths represented by the attack tree, according to the semantics of Mauw and Oostdijk (2006): {{employee takes laptop, employee locks door, employee moves to hallway, employee leaves laptop, take laptop}, {enter room, employee locks door, employee moves to hallway, take laptop}, {employee locks door, employee moves to hallway, employee leaves key, take key, unlock door, enter room, take laptop}}.

In the ANKH analysis, the additional obligations associated with these failure cases can also be added to the policies of the hypergraph entities, in order to evaluate them. This prevents the corresponding actions (graph transformations) from occurring, and thus removes the corresponding branches from the attack tree (Fig. 6). For the first additional obligation, the node "employee leaves room" gets an additional child "attacker not in room". This adds a sequence 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 "employee leaves laptop" and "employee leaves key", removing the possibility that either the key or the laptop is left outside. Similarly to the InstAL analysis, adding both obligations thereby blocks all paths.

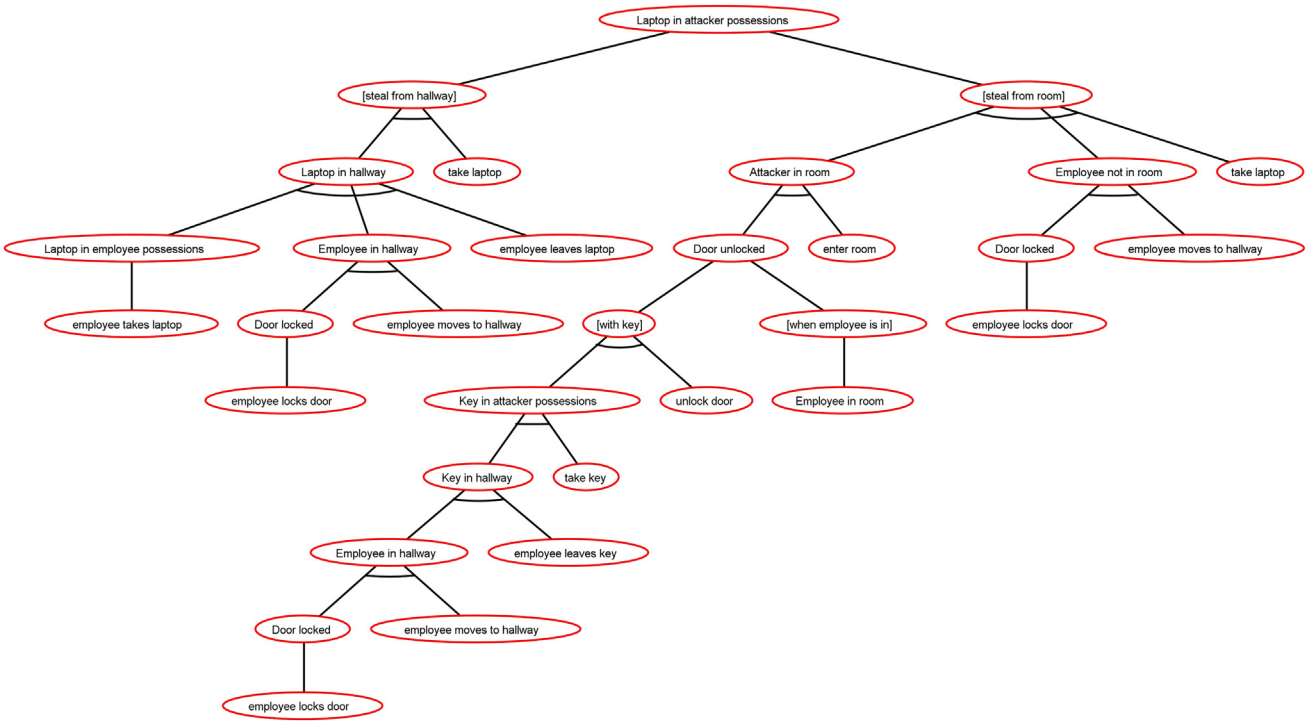


Fig. 6 – The attack tree of the example in ADTool format. Nodes with an arc underneath are AND-nodes, all other nodes are OR-nodes. Next to the preconditions (capitalised), explicit actions (without capitals) are present as children of a postcondition. Preconditions that were already true in the initial state are omitted, as these are not part of the attack. Nodes in brackets serve as intermediary nodes between AND- and OR-gates, which cannot be directly connected in ADTool.

Later in this paper, we will also consider the possibility of adding these additional policies explicitly to the tree.

6. Quantitative models

6.1. *Time annotations*

Instead of requiring that certain actions are impossible for an attacker, which is a binary property, we can consider the question what the analysis would look like if we require that certain actions have at least a certain level of *difficulty* for an attacker. We could then provide a quantitative analysis of the relation between obligations and prohibitions.

Difficulty for the attacker can be expressed, for example, as the time, effort, or money required to perform an attack, or an attack step. In this paper we choose time, as time is more amenable to measurement than for example money. In a different line of research, we consider multi-parameter situations, including constraints on attacker capacity (Pieters et al., 2014). Assume that entering a room costs 10 s when the door is open, 20 s when the door is locked but the person has the key, and 3 min when the door is locked and the person does not have the key (but presumably some burglary tools). Attack trees can be annotated with such values, representing the time required for the atomic actions. From those, times required for composite attacks can be calculated.

Such calculations are based on an attacker point of view. When time annotations are present as annotations in an attack tree, optimal solutions can be calculated from the point of view of the attacker. For example, if the attacker can choose between two paths (OR-node), he will choose the path with the minimal time, and thus this minimal time is associated with the goal represented by the OR-node. When the attacker has to perform multiple actions to reach a goal (AND-node), the sum of the times will be associated with the goal of the AND-node. A special infinity value (∞) may be used to indicate that an action is not possible at all.

In this way, we can add all times associated with actions in a path that leads to violation of the prohibition on the attacker.

6.2. *Quantitative policies*

The next step is relating such quantitative information to security policies. It is one thing to associate a required time with a certain behaviour or a certain goal, but how to express prohibitions and obligations? In principle, there is not much difference in associating a time with such policies, except that the time is now interpreted as a *minimum* time. That is, if a policy specifies that a certain goal has a certain time, this means that it should take an external agent *at least* this amount of time to reach that goal. For example, it should take an attacker at least 3 min to steal a laptop. Such time-based policies are already in use in burglary prevention.[3]

[3] http://tinyurl.com/oxjyad9, consulted January 27, 2013.

Again, this prohibition on external agents can be translated into an obligation for employees. Thus, the obligations on the employees would define a minimum level of resistance they should provide when an attacker attempts to perform a certain attack step. On a high level, one could state that the employees should see to it that stealing a laptop costs an attacker at least 3 min. On a lower level, one could state that the employees should see to it that obtaining the key would cost the attacker at least 2 min. The latter policy would then contribute to the realisation of the former. Again, the value ∞ can be used to state that a particular attacker action should not be possible at all.

We formalise this by adding attribute-value pairs to policies. We assume that values in each attribute domain are partially ordered. With a representing a set of attribute-value pairs, policies would now look as follows:

- $F_x\phi(X,a)$, a prohibition on the attacker to achieve ϕ with any of the attributes in a having a value $\leq$ the value specified in a (e.g. in less than 3 min);
- $O_E STIT_E(\neg\phi(X,a))$ an obligation and responsibility on the employees to see to it that an attacker cannot achieve what has been prohibited with any of the attributes in a having a value $\leq$ the value specified in a (e.g. in less than 3 min).
- obligations on agents to see to it that certain actions have attribute values such that specified minimum levels for the attributes are $\leq$ these values.

Note that the above reasoning with required time carries implicit assumptions on the skill of the attacker. Here, we assume that those specifying the policies do have reasonable assumptions on this for a particular type of attacker. It would be possible to specify explicit attacker profiles for this purpose, performing analyses for attackers with different skill levels (Arnold et al., 2013), but we do not discuss this issue here.

7. Quantitative analysis

Based on quantitative annotations of actions and quantitative policies, we can analyse again in ANKH whether a certain security policy is satisfied, i.e. whether the obligation to prevent the attacker from reaching his goal is implied by the obligations with respect to atomic actions. To this end, we extend policies in the ANKH model with attribute-value pairs, which are then transferred to the generated attack trees. As this only requires annotations on the possible attack steps, it does not affect the theoretical worst-case complexity. However, actions that may have been considered impossible in the original model may now be considered very costly instead. This means that these actions have to be included in the analysis, yielding a higher expected running time.

7.1. *Quantitative policies*

Remember that in ANKH, we represent access policies as tuples (g,e,C,S,T), and meta-policies as (P,C). For the access policies, we can include the quantitative values in which we are interested in the policy. Formally, a quantitative policy is represented as a tuple (p,a), with p an access policy, and a a set of attribute-value pairs.

For the meta-policy, we have to take into account the possibility that the quantitative values differ based upon the credential used. For example, it may take more time to unlock a door with a key from the outside than it takes to unlock the same door with a knob from the inside. Thus, a quantitative meta-policy is a tuple (P,V), with P a set of policies, and V a set of tuples (C,a), with C a set of credentials and a a set of attribute-value pairs. This allows us to express quantitative annotations for both complying with the active policy, and for changing the active policy by complying with the meta-policy.

7.2. *Attack tree generation and analysis*

Again, we can generate an attack tree for the example. The tree is structurally equivalent to the original tree. The difference is that the attribute-value pairs from the policies are transferred into the tree. For example, in node "enter room", the attacker acquires access to the room with the key. As the associated policy (the one stating that the key gives access to the room) now has a set of attribute-value pairs attached, these will also be attached to this node in the attack tree. For example, this node may now state that the time required is 30 s. Formally, the set of attribute-value pairs {(*time*,0.5)} is attached (with time in minutes).

Analysis of quantitative annotations of attack trees is a well-established area (Buldas et al., 2006; Kordy et al., 2012; Mauw and Oostdijk, 2006). In addition, there is tool support available for this purpose (Kordy et al., 2013). Therefore, it would be beneficial if we could leverage this existing work for the analysis of the attack trees generated by the analysis. In particular, for time annotations, a parent node will acquire as a time value (a) the *sum* of the time values of its children, if it is an AND-node (assuming sequential execution of the children); and (b) the *minimum* of the time values of its children, if it is an OR-node (assuming that the attacker prefers the fastest option). Based on this rule, one can calculate the time value of the root node.

To show the practical applicability, we demonstrate the use of the quantitative graph-based analysis of policies by means of ADTool (Kordy et al., 2013). ADTool enables development and analysis of quantitative attack trees and attack-defense trees. The annotated attack tree in ADTool is shown in Fig. 7, the annotations representing required time in minutes.

Assume we have a quantitative policy stating that it should not be possible for an attacker to obtain a laptop in less than 3 min. As shown in the calculations in the tree, the required time is now only 1.5 min (annotation of the root node). The root node is an OR-node, with the children having 2.5 and 1.5 min as annotations, respectively. This means that, in order to get the overall required time up to 3 min, both branches will have to change. Relating to the solutions in the qualitative case, we can again propose additional obligations:

- see to it that it takes an attacker at least a certain amount of time to find a laptop in the hallway;
- see to it that it takes an attacker at least a certain amount of time to find a key in the hallway;

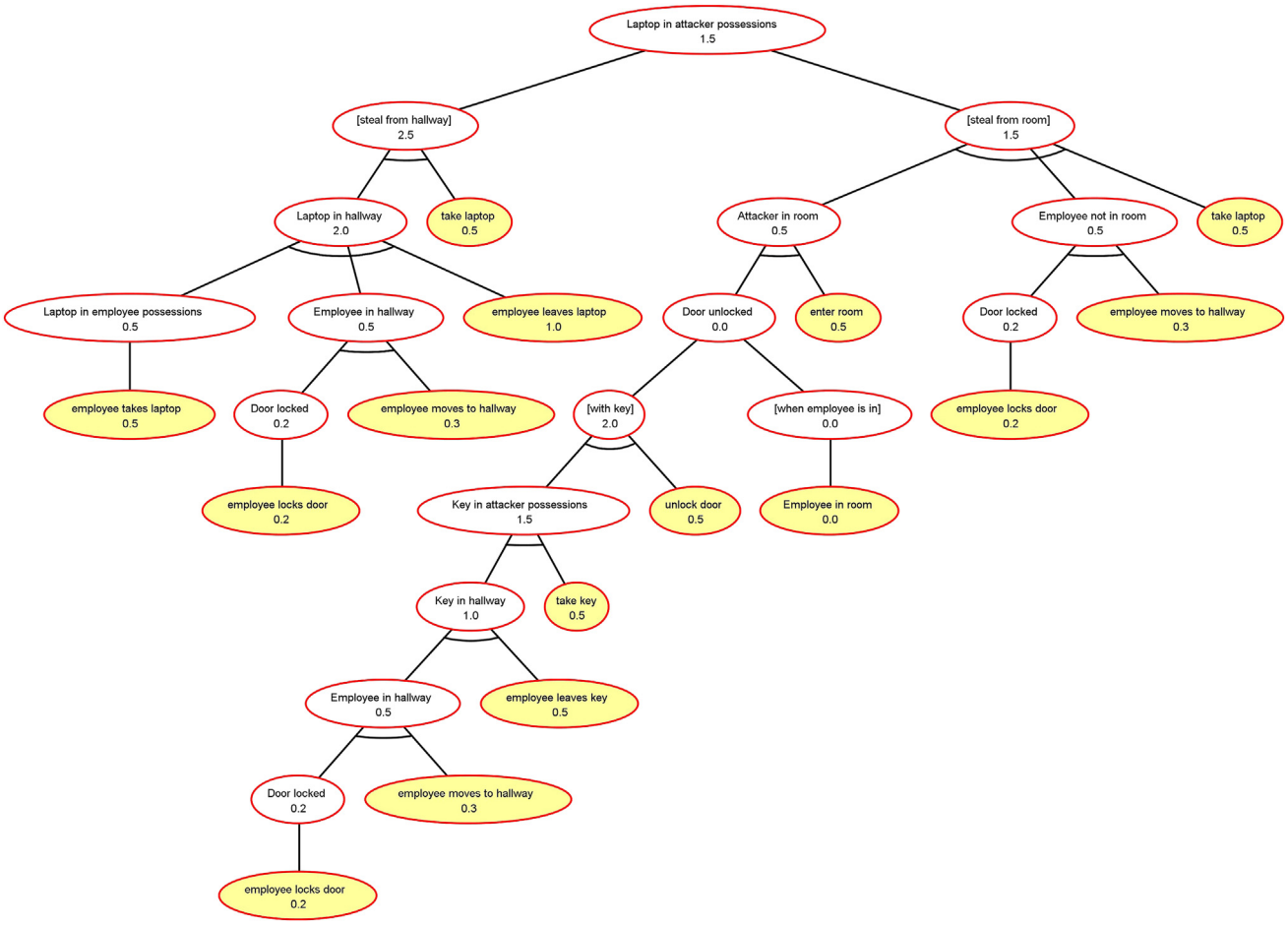


Fig. 7 – The attack tree of the example in ADTool with time annotations in minutes. The leaf nodes contain values that have been supplied by the user; the values in the non-leaf nodes have been calculated by the tool. It is assumed that the actions have to take place sequentially; therefore, AND-nodes will be annotated with the sum of the annotations of their children.

- see to it that employees do not lock and leave the room while somebody else is there.

In the ANKH model, these obligations will increase the time associated with policies in the model. When an additional policy makes an action impossible (as in the third case), the time will increase to ∞. These values will then be propagated to the nodes in the tree. The last obligation addresses a dependency between different parts of the tree: the attacker entering the room and the employee leaving. Standard attack trees are not particularly suitable to express this, and for this particular obligation, more advanced methods such as Bayesian networks (Poolsappasit et al., 2012) might be more appropriate, in order to take the dependencies into account. We are currently investigating this option.

7.3. *Effectiveness of policies*

If we choose required time as annotation for attack scenarios, we can say that the *marginal effectiveness* of an additional policy (could be a defense node) is the increase in required time for the associated attacker goal. As can easily be seen from attack tree analysis (in which the time of an OR-node is the minimum of the time of the children) additional policies will only be effective if they change the value of actions along the path with the lowest value. For different annotations (e.g. probability of success), the definition of marginal effectiveness/adequacy is analogous. In the boolean case (qualitative analysis), one could state that an additional policy (or set of additional policies) is only effective if it blocks all possible attack paths, i.e. changes the possibility of reaching the goal from true to false.

In addition, we can express the degree of compliance to a policy as the ratio between the actual time an attacker needs and the required time. In the example, this degree would be $1.5/3 = 1/2$.

8. Further extensions

8.1. *Attack steps and attack events*

Note that there is an additional complication here, which has to do with the distinction between actions that are controlled by the attacker and actions that are not controlled by the attacker. For example, opening the door with a key is an action

that is controlled by the attacker, but the employee leaving the laptop in the hallway is not controlled by the attacker. We call the former *attack steps* and the latter *attack events*. If we annotate both types of actions with times, these annotations have slightly different meanings. The time required to open a door specifies the *time spent by the attacker*, whereas the time until the employee leaves the laptop in the hallway specifies the *real time elapsed until an event* (i.e., mean time to failure). This distinction could be explored further in future work.

Additional properties that may be considered in future work are the probability of detection (opening a door by force causes noise, which is a risk for the attacker), and probability of success. The probability of detection may in turn depend on the time invested by the attacker, such that the annotation would be a functional relation between properties rather than just a set of independent values.

8.2. *Explicit defense nodes*

Explicit defense nodes allow for inclusion of defenses as separate entities in attack trees, rather than by their influence on values of attack nodes. For example, providing a training for employees could increase the expected time required for an attacker to find the laptop in the hallway, but one could also model this with a defense node attached to the attack node.

In particular, going all the way back to the link between prohibitions and obligations introduced in the beginning of this paper, we could reason as follows:

1. There is a prohibition on external agents to obtain possession of a laptop within a certain amount of time (say 3 min);
2. There is a general obligation on all employees to see to it that an external agent cannot obtain possession of a laptop in less than 3 min;
3. This general obligation can be translated into specific obligations, ensuring that specific steps contributing to the goal of obtaining possession of the laptop take at least a certain amount of time;
4. If the analysis yields attack paths that take less than 3 min, additional defenses (policies) are needed that increase the time for some of the steps along these paths;
5. Such defenses/policies could be represented as defense nodes, provided that these would *add* a certain amount of time to the time originally required for the step.

For example, if finding a laptop in the hallway would originally require 2 min, and this would yield an attack path of 2.5 min, an awareness campaign would be sufficient if it would increase the time required for finding a laptop by 30 s to 2.5 min. Or if an attacker can obtain a laptop by entering an open room, with entering the room taking 30 s and the overall attack 2 min, then a policy requiring the door to be locked would be effective under certain conditions. In this case, opening the door with a key would take 30 s (30 extra seconds compared to an open door), and opening by force would take 3 min. Therefore, the policy would be effective if finding the key would take an attacker at least 30 s, such that the overall increase would be 30 + 30 s, bringing the overall time to 3 min.

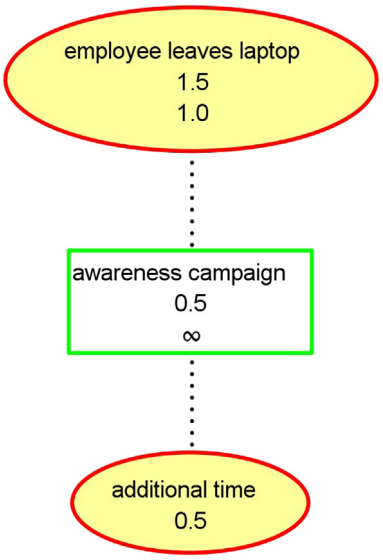


Fig. 8 – An obligation for employees to increase the required time for the attacker as a defense node.

For these examples, time annotation ∞ may indicate that employees are obliged not to leave laptops or keys in the hallway at all, basically supporting qualitative policies in the quantitative model.

The *additional* time imposed by the additional policies could thus be represented as a defense node. However, in ADTool, such a defense node would block the attack completely, unless there is a counterattack on the defense node. For example, an awareness training node would block finding the laptop in the hallway, unless the attacker would counterattack this defense with another attack. Therefore, we would need to model the additional time required on the part of the attacker as a *counterattack on the defense node*, where this counterattack would be annotated with the additional time required (Fig. 8). In this way, we can use the existing semantics of defense nodes in order to represent countermeasures as *additive* for the time required, and they could thus be used to represent specific (additional) obligations for employees. Ensuring that these obligations are met can be achieved by specific defenses such as awareness campaigns.

9. Conclusions

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 obligations 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 logic-based and graph-based approaches can be used to execute the analysis, with the graph-based approach having the advantage that it supports quantitative policies and quantitative analysis, for example in relation to the time required for the

attack. As outlined above, the framework can be employed to check the completeness of obligation policies put on employees, with respect to the goal of preventing undesired states, or, in a quantitative setting, making sure that the attribute-value pairs meet the required constraints. We will investigate the possibility to extend the logic-based model to support such attributes as well.

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 $\mathscr{E}$) 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.

Compared to existing work, we provide two main contributions. Firstly, we explicitly introduce obligations into security models, thereby extending the possibilities for modelling socio-technical aspects. Existing models such as Portunes (Dimkov, 2012), ExASyM (Probst and Hansen, 2008), ANKH (Pieters, 2011), CySeMoL (Holm et al., 2013) and CORAS (den Braber et al., 2007) do not explicitly address obligations. Furthermore, although analysis frameworks do support quantitative analysis (Buldas et al., 2006; Kordy et al., 2012; Poolsappasit et al., 2012), this analysis is not linked to quantitative policies. Quantitative policies have so far only been addressed in a technical setting (Degano et al., 2011). Reasoning with quantitative policies in socio-technical security settings is therefore our second main innovation.

One disadvantage of quantitative policies would be the increased complexity of policy management. In particular, it will be more difficult to judge without tool support whether policies are consistent, which may inhibit incentives for adoption. Also, the policies will be harder to understand, leading to potential usability concerns. However, as quantitative policies are already successfully used in physical security, we think that with adequate further research and tool support they have significant potential for cyber security as well.

In future work, we will focus on including support for step-wise refinement of the prohibitions for attackers into obligations for employees. In addition, we will investigate the incorporation of quantitative attributes for obligations in the logic-based model. We also aim at including explicit attacker profiles in the quantitative graph-based analysis, to account for the fact that the required time depends on both attacker and defender properties. Based on this approach, other attributes may be added as well.

Acknowledgments

The research leading to these results 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 under project 09.08.KID.

REFERENCES

Abrams M, Bailey D. Abstraction and refinement of layered security policy. In: Information security: an integrated collection of essays. IEEE Computer Society Press; 1995. p. 126–36.

Arnold F, Pieters W, Stoelinga M. Quantitative penetration testing with item response theory. In: 9th International Conference on Information Assurance and Security (IAS 2013). IEEE; 2013.

Baral C. Knowledge representation, reasoning and declarative problem solving. Cambridge University Press; 2003.

Buldas A, Laud P, Priisalu J, Saarepera M, Willemson J. Rational choice of security measures via multi-parameter attack trees. In: Lopez J, editor. Critical information infrastructures security. Lecture Notes in Computer Science, vol. 4347. Berlin Heidelberg: Springer; 2006. p. 235–48.

Cholvy L, Cuppens F. Analyzing consistency of security policies. In: Security and Privacy, 1997 IEEE Symposium; 1997. p. 103–12.

Cholvy L, Cuppens F, Saurel C. Towards a logical formalization of responsibility. In: Proc. of the 6th int. conf. on Artificial intelligence and law, ICAIL '97. New York, NY, USA: ACM; 1997. p. 233–42.

Cliffe O, De Vos M, Padget J. Answer set programming for representing and reasoning about virtual institutions. In: CLIMA VII. Lecture Notes in Computer Science, vol. 4371. Springer; 2006. p. 60–79.

Degano P, Ferrari G-L, Mezzetti G. On quantitative security policies. In: Malyshkin V, editor. Parallel computing technologies. Lecture Notes in Computer Science, vol. 6873. Berlin Heidelberg: Springer; 2011. p. 23–39.

den Braber F, Hogganvik I, Lund M, Stølen K, Vraalsen F. Model-based security analysis in seven steps – a guided tour to the CORAS method. BT Technol J 2007;25(1):101–17.

Dimkov T. Alignment of organizational security policies – theory and practice [PhD thesis]. Enschede: University of Twente; February 2012.

Dimkov T, Pieters W, Hartel P. 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); Dec. 2010. p. 727–32.

Feltus C, Petit M, Sloman M. Enhancement of business it alignment by including responsibility components in RBAC. In: Proceedings of the CAiSE 2010 Workshop Business/IT Alignment and Interoperability (BUSITAL2010); 2010. p. 61–75.

Gelfond M, Lifschitz V. Action languages. Electron Trans Artif Intell 1998;2:193–210.

Holm H, Sommestad T, Ekstedt M, Nordström L. CySeMoL: a tool for cyber security analysis of enterprises. In: Electricity Distribution (CIRED 2013), 22nd International Conference and Exhibition. IET; 2013. p. 1–4.

Hopton L, Cliffe O, De Vos M, Padget J. AQL: a query language for action domains modelled using answer set programming. In: Erdem E, Lin F, Schaub T, editors. LPNMR. Lecture Notes in Computer Science, vol. 5753. Springer; 2009. p. 437–43.

Kammüller F, Probst CW. Invalidating policies using structural information. In: Proceedings of 2013 IEEE Security and Privacy Workshops; 2013. p. 76–81.

Kordy B, Kordy P, Mauw S, Schweitzer P. ADTool: security analysis with attack-defense trees. In: Joshi K, Siegle M, Stoelinga M, D'Argenio PR, editors. Quantitative evaluation of systems. Lecture Notes in Computer Science, vol. 8054. Berlin Heidelberg: Springer; 2013. p. 173–6.

Kordy B, Mauw S, Radomirovic S, Schweitzer P. Attack-defense trees. J Logic Comput 2012. http://dx.doi.org/10.1093/logcom/exs029.

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

Mauw S, Oostdijk M. Foundations of attack trees. In: Information Security and Cryptology – ICISC 2005. Lecture Notes in Computer Science, vol. 3935. Springer; 2006. p. 186–98.

Meyer J-JC, Wieringa RJ. Deontic logic: a concise overview. In: Meyer J-JC, Wieringa RJ, editors. Deontic logic in computer science: normative system specification. Chichester, UK: John Wiley & Sons; 1993. p. 3–16.

Nunes Leal Franqueira V, van Eck PAT. Towards alignment of architectural domains in security policy specifications. In: Parente de Oliveira JM, Westphall CB, Brustoloni JC, editors. Proc. 8th Int. Symp. System and Information Security. Fundacao Casimiro Montenegro Filho – CTA/ITA; 2006.

Pieters W. Representing humans in system security models: an actor-network approach. J Wirel Mob Netw Ubiquit Comput Depend Appl 2011;2(1):75–92.

Pieters W, Dimkov T, Pavlovic D. Security policy alignment: a formal approach. Syst J, IEEE 2013a;7(2):275–87.

Pieters W, Hadziosmanovic D, Lenin A, Montoya L, Willemson J. TREsPASS: plug-and-play attacker profiles for security risk analysis. In: IEEE Security & Privacy poster abstracts. IEEE; 2014. http://www.ieee-security.org/TC/SP2014/posters/PIETE.pdf.

Pieters W, Padget J, Dechesne F, Dignum V, Aldewereld H. Obligations to enforce prohibitions: on the adequacy of security policies. In: Proceedings of the 6th International Conference on Security of Information and Networks, SIN '13. New York, NY, USA: ACM; 2013b. p. 54–61.

Poolsappasit N, Dewri R, Ray I. Dynamic security risk management using Bayesian attack graphs. IEEE T Depend Secure 2012;9(1):61–74.

Probst CW, Hansen RR. An extensible analysable system model. Inform Security Tech Rep 2008;13(4):235–46.

Sloman M. Policy driven management for distributed systems. J Netw Syst Manage 1994;2(4):333–60.

Sloman M, Lupu E. Security and management policy specification. Netw, IEEE 2002;16(2):10–9.