Security Policy Alignment: A Formal Approach

Wolter Pieters, Trajce Dimkov, and Dusko Pavlovic, *Member, IEEE*

***Abstract*—Security policy alignment concerns the matching of security policies specified at different levels in socio-technical systems, and delegated to different agents, technical and human. For example, the policy that sales data should not leave an organization is refined into policies on door locks, firewalls and employee behavior, and this refinement should be correct with respect to the original policy. Although alignment of security policies in socio-technical systems has been discussed in the literature, especially in relation to business goals, there has been no formal treatment of this topic so far in terms of consistency and completeness of policies. Wherever formal approaches are used in policy alignment, these are applied to well-defined technical access control scenarios instead. Therefore, we aim at formalizing security policy alignment for complex socio-technical systems in this paper, and our formalization is based on predicates over sequences of actions. We discuss how this formalization provides the foundations for existing and future methods for finding security weaknesses induced by misalignment of policies in socio-technical systems.**

***Index Terms*—Attack trees, security logics, security policies, security policy alignment, security policy refinement, socio-technical systems, system models.**

I. Introduction

COMPLEXITY in socio-technical systems is increasing. Systems composed of information, physical properties and human behavior have always been sophisticated, but recent developments make a real difference. Outsourcing and service composition cause dissolution of boundaries between organizations. The proliferation of mobile devices causes dissolution of boundaries between the private and the public sphere, between work and home. Convergence of access control mechanisms, as well convergence of bio-, nano- and info-technologies cause dissolution of boundaries between different technologies. These trends lead to an explosion of the number of possible interactions.

When considering the security of information in such socio-technical systems, developments such as working from home, bring-your-own-device, and cloud computing lead to increasingly complicated information security problems. One has to deal with propagation of access rights in complex attack scenarios: attackers may exploit vulnerabilities at different levels, and attacks may include physical access and social engineering. This is already the case even in relatively simple scenarios. For example, in the road apple attack, an attacker will leave infected dongles around the organization's premises. When an employee picks up a dongle and plugs it into a company computer, malware will send out all the information that it can find. The possibilities for such multistep attacks in increasingly complex systems come with the important questions of how to manage information security policies in complex situations, and how to check whether the security policies in place are adequate.

The question of adequacy of security policies (i.e., whether existing policies provide sufficient protection against the targeted threats) can be addressed from the perspective of security policy alignment. Security policies may be stated at different levels of abstraction, where higher-level policies are refined into lower-level policies. For example, the policy that sales data should not leave the organization is refined into policies on door locks, firewalls, and employee behavior. In security policy alignment, security policies are tested against each other, and against business goals, to determine whether they match the associated constraints. Informal approaches to assess security policy alignment already exist (e.g., [1]–[4]). However, as in many other socio-technical aspects of information security, a formalization of the concepts lacks. On the other hand, where formal approaches are used, these are often limited to fairly simple logical access control problems [5]–[8]. In these frameworks, permission is discussed in terms of single actions, but not sequences of actions. This reduces the value of the notion of policy alignment in inspiring formal analysis of information security in complex systems, where multistep attack scenarios are typical. Also, the relation between policy alignment, security logics, and model checking approaches remains unclear.

Solhaug and Stølen [9] do discuss policies in terms of sequences of actions. Their framework allows refinement of both systems and policies, based on UML specifications. However, they do not explicitly address security, and therefore

Manuscript received October 7, 2011; revised July 11, 2012; accepted July 26, 2012. Date of publication November 29, 2012; date of current version April 17, 2013. Part of this research was done when W. Pieters and T. Dimkov were affiliated with the VISPER Project, University of Twente, Enschede, The Netherlands. This research was supported by the Research Program Sentinels (www.sentinels.nl). Sentinels is being financed by Technology Foundation STW, The Netherlands Organization for Scientific Research, and the Dutch Ministry of Economic Affairs. The research of W. Pieters is currently supported by financial assistance from the European Commission in the context of the SESAME Project. The views expressed herein are those of the authors and can, therefore, in no way be taken to reflect the official position of the European Commission.

W. Pieters is with the Energy and Industry Group, Faculty of Technology, Policy, and Management, Delft University of Technology, Delft 2600, The Netherlands (e-mail: w.pieters@tudelft.nl).

T. Dimkov is with the Security and Privacy Group, Deloitte, Amstelveen 1183, The Netherlands (e-mail: tdimkov@deloitte.nl).

D. Pavlovic is with the Information Security Group, Royal Holloway, University of London, Egham TW20 0EX, U.K., and also with the Distributed and Embedded Security Group, Faculty of Electrical Engineering, Mathematics, and Computer Science, University of Twente, Enschede 7500, The Netherlands (e-mail: dusko.pavlovic@rhul.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2012.2221933

do not support essential concepts in this field. In particular, we need notions of completeness of policies, and attacks in the case of incompleteness, and the notion of policies on system states rather than traces.

To resolve this situation, and make the connection between the different approaches explicit and precise, we provide a formalization of security policy alignment in arbitrary types of (socio-technical) systems, by providing mathematical definitions of the central concepts, and the relations between those. This can be seen as an effort complementary to the formalization of attack trees by Mauw and Oostdijk [10] and attack-defense trees by Kordy, Mauw, Radomirović, and Schweitzer [11]. Whereas attack trees represent possible undesirable behaviors, they do not contain an explicit notion of policy or permitted behavior. This extension of our formal understanding is necessary to reason about the matching between different policies in a system, and the relation between policy mismatches and attack scenarios.

Our approach focuses on the expression of security policies at different levels of abstraction, specifying whether behaviors are permitted or forbidden, and the consistency and completeness of such policies with respect to policies at other levels. We define policies on traces (sequences of actions) rather than individual actions such that both high-level policies (sales data should not leave the organization) and low-level policies (this door can only be opened with a specific key) can be expressed in the same framework. We make a distinction between local and global constraints on traces (see Section IV). Traces are generated from a system model, which we leave implicit until Section VI. Existing socio-technical system models can be plugged in for this purpose [12]–[14].

We make no distinction between descriptive and normative policies (e.g., "the door can only be opened with the key" versus "it should only be allowed to open the door with the key"), as this is only a matter of abstraction: what is normative on a high level is implemented by descriptive policies at a lower level, and when these lower level policies are further refined, they become normative in turn. This allows us to express policies in a relatively simple framework. Similarly, security mechanisms are seen as low-level security policies, and, indeed, such low-level policies can enforce multiple high-level policies, and there may be several possible low-level policies that enforce the same high-level policy [15].

Although the main aim of this paper is theoretical, in the sense that we provide formal foundations for policy alignment, it has substantial practical implications in terms of connecting existing methods for security analysis, and in providing opportunities for future applied research in this area.

In Section II, we introduce the concepts involved in security policy alignment, and a running example. In Section III, we provide the basic formal definitions, including consistency of policies, completeness and soundness of policies, and their relations. In Section IV, we distinguish between different types of policies, which has practical consequences for methods of analysis. Model checking consistency and completeness is discussed in Section V, with procedures to generate attacks from mismatches between global policies and local ones highlighted in Section VI. In Section VII, we discuss possible applications of the framework, followed by related work (Section VIII) and conclusion (Section IX).

II. Security Policy Alignment

Organizations protect sensitive information by means of describing and implementing security policies. Policies can be defined at different levels of abstraction. High-level policies describe the assets of the organization, as well as desirable and undesirable states of such assets (e.g., in the hands of competitors). Human resources (HR), physical security, and IT departments refine these policies into implementable, low-level policies [16], which are enforced via physical and digital security mechanisms and training of the employees. These policies describe the desired behavior of the employees (social domain), the physical security of the premises where the employees work (physical domain), and the IT security of the stored and processed information (digital domain) [17] such that these refinements together realize the high-level policies.

During the refinement and enforcement of the policies, mistakes may occur. These mistakes could be exploited by both external parties and insiders [18] to achieve a malicious goal. Therefore, management needs assurance that both refinement and enforcement are done correctly. This assurance is achieved in two steps: auditing and penetration testing. During the auditing process, auditors assess whether the security policies produced by the departments are correct with respect to the policies defined by the management. After the policies from the departments have been audited, penetration testers test whether the security mechanisms correctly enforce the policies from the departments.

The current work focuses on the auditing of security policies by comparing formalized security policies in socio-technical systems (e.g., organizations) and systematically checking the refinement of high-level into low-level policies. We use the informal description of policy alignment, as presented by Abrams, Olson, and Bailey [19], [20] as a basis. The definitions in this section provide informal intuitions for the reader's convenience; formal definitions will be provided in the next section.

Definition 1: Security policy alignment is the process of adjusting to each other different security policies for a system.

When considering a single level of abstraction, consistency of the policies for a system is the most important concern in policy alignment. When considering multiple levels, we speak of policy refinement.

Definition 2: Security policy refinement is the process of defining policies with a greater level of detail to support a given general security policy.

This definition does not say anything about whether the refinement is correct; requirements for correctness will be discussed and formalized in the following.

The refinement step should be repeated for each level of abstraction, starting from the policies defined on the highest level of abstraction, toward policies on a lower level of abstraction [19]. In refinement, completeness of lower-level policies with respect to the original policy is an important concern. Moreover, lower-level policies should again be mutually consistent. To simplify the presentation, we use just two levels of abstraction for the policies, which we call high-level and low-level policies. High-level policies are focused on security goals with respect to the assets of the organization ("sales data should not leave the organization"), and low-level policies constrain individual actions of actors ("this door can only be opened with a specific key").

We do not focus on the problem of translating policies from natural language into formal languages. The examples are intuitive enough for explanation purposes, and the translation of policies from natural to formal languages is a research topic by itself [21]. An example of the interpretation problems that can occur is provided in [4, ch. 5]. To what extent such a process can be automated remains to be seen. In the framework presented in this paper, most policies are relatively simple access relations (should not have access to..., will grant access to...), and therefore we believe the translation problems are manageable.

Policies are specified in terms of permitted or forbidden behaviors. A behavior is a sequence of actions, where an action is a discrete event that cannot be broken up further. Policies divide the space of possible behaviors into behaviors that are permitted, behaviors that are forbidden, and behaviors that are neither forbidden nor permitted. For high-level policies, the set of behaviors may not even be specified yet, as high-level policies are often stated in terms of all behaviors that have an undesirable outcome. For example, a high-level policy may state that all behaviors leading to the sales data ending up outside the organization are forbidden, without specifying what exactly these behaviors are. Depending on the refinement of the system into components [22], [23], it will then become possible to tell which behaviors actually lead to the undesirable outcome.

When a system is specified in more detail, either by designing the system or by empirically investigating it, the policies will also reappear at lower levels. Such low-level policies are policies that are delegated to system components, such as doors, firewalls, and humans. Rather than specifying what is permitted or forbidden depending on the outcomes of a behavior, low-level policies typically permit or forbid actions based on the executing agent, the location, and/or the credentials. Also, low-level policies are typically more exhaustive in the sense that more behaviors will be explicitly forbidden or permitted than in the high-level policies. In this way, the number of behaviors with unspecified permission will be reduced. When low-level policies are specified in terms of individual actions rather than complete behaviors, they still apply to behaviors: a behavior is allowed by the low-level policies if all the actions of which it consists are allowed by the low-level policies, and a behavior is forbidden by the low-level policies if at least one of its actions is forbidden by the low-level policies.

In this paper, we assume actions to be atomic events on a single level of abstraction. Although actions can be specified at different levels when discussing system refinement, for security policy alignment we are interested in the refinement of the policies, not the actions. This refinement occurs primarily in terms of different levels of policies, namely, policies that refer to the actions themselves, and policies that refer to the outcome of actions. In this context, stating that a certain behavior or outcome is not permitted is equivalent to stating that the corresponding sequences of actions are not permitted on the chosen level of abstraction. The translation of actions into a single level of abstraction will not be discussed further here.

As an example of policy alignment, suppose that an organization has a high-level policy that enforces a behavior: "aggregate sales data should be given to all shareholders." With the introduction of a policy that forbids a behavior, "sales data should not leave the financial department," the set of high-level policies is no longer consistent. There is a conflict between the two policies because the second policy forbids the sales data leaving the financial department, while the first policy requires some of the sales data to leave the organization. This is an example of misalignment by inconsistency.

A high-level policy might also be refined into overly permissive or overly restrictive low-level policies, which introduces an opportunity for an adversary to violate the high-level policy by means of an attack. We consider attacks as sequences of actions that conform to a refined set of policies, while violating the corresponding higher-level policy.

Example 1: As a running example, we consider a variant of the road apple attack [24]. This attack consists of the following sequence of actions.

1) The attacker prepares dongles with malware and company logo.
2) The attacker places dongles in a publicly accessible location (say canteen).
3) The employee takes one dongle and plugs it into a computer.
4) Autorun installs rootkit on the computer.
5) Rootkit acquires sales data.
6) Rootkit encrypts sales data.
7) Rootkit sends encrypted sales data out (firewall permits encrypted egress traffic).
8) The attacker receives encrypted sales data.

A high-level policy of the organization states that sales data should not leave the organization. If all of the above actions are possible, they constitute a violation of the high-level policy. In this example, overly permissive low-level policies, such as allowing employees to bring storage devices to work and allowing dongles to be plugged in the computer, allow the violation of the high-level policy. Thus, there is a misalignment of policies by incompleteness.

This general overview provides the most important intuitions for our approach to policy alignment. We will define the associated notions of action, behavior, and policy more precisely in the following.

III. Formal Definitions

In order to formalize policy alignment, we consider the concepts of action, behavior, and policy. Then, we define policies as first-order logic theories with permission predicates over behaviors. We will define alignment in terms of consistency

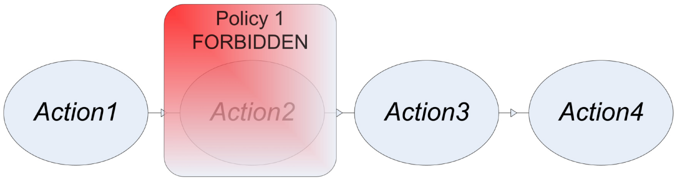


Fig. 1. Visualization of prefix closedness of the permission predicate. If a policy would forbid Action2 in the context of a preceding Action1, it would effectively forbid all behaviors with prefix (Action1, Action2). However, Action2 might be permitted when, say, Action5 would precede it instead.

and completeness of policies. Our notation is similar to that in [25].

Consider a set of abstract atomic actions $\mathcal{E}$ (for events). We will call sequences of actions behaviors, with ϵ denoting the empty behavior.[1] The set of all possible behaviors is denoted $\mathcal{T} = \mathcal{E}^*$ (for traces). For a behavior $T \in \mathcal{T}$, we use the predicate $P(T)$ to indicate that T is permitted.

Definition 3: A policy is a theory Θ in first-order logic, with behaviors $T \in \mathcal{T}$ being the terms and $P(\_)$ a distinguished prefix closed predicate over behaviors.

The formula $P(T)$ means that behavior T is permitted or possible; $\neg P(T)$ means that a behavior T is forbidden or impossible. If neither $P(T)$ nor $\neg P(T)$ can be derived from a policy, then the permissibility of T is undecided. For example, the policy $\{\neg P(Action1, Action2)\}$ would forbid all behaviors beginning with $Action1$ followed by $Action2$. More complex formulae and sentences can be built using standard first-order logic constructs. A policy is a theory and thus consists of a set of sentences.

With prefix closed, we mean that for every behavior T and action e, $P(Te) \to P(T)$, with Te representing the behavior T extended with action e. If a policy forbids a behavior, it should also forbid any behaviors that extend this behavior. Similarly, if a policy allows a behavior, it should also allow its prefixes (see also Fig. 1). Prefix closedness implies that certain types of theories will not be policies. For example, for any behavior T and action e, the theory $\{P(Te), \neg P(T)\}$ will not be possible with a prefix closed predicate P, and cannot serve as a policy. Thus, a theory that allows an employee to enter a room and then pick up a dongle, but forbids the said employee to enter the said room, is not a policy.

We say that a policy is consistent if no contradictory formulae can be derived from it.

Definition 4: A policy Θ is consistent iff there is no formula ϕ such that $S \vdash \phi$ and $S \vdash \neg\phi$.

A new policy can be formed from the union of a set of policies, that is $\Theta' = \bigcup_{i=1}^{n} \Theta_i$. When the new policy Θ' is consistent, we say that $\Theta_1 ... \Theta_n$ are mutually consistent.

Policies can be represented in Venn diagrams of the space of behaviors, where for each behavior T it is indicated whether T is permitted, forbidden, or undecided. When representing multiple policies in the same diagram, one can visualize possible contradictions. For mutually consistent sets of policies, the space can be divided into permitted, not permitted, and unspecified. A mutually inconsistent pair of policies is shown in Fig. 2.[2]

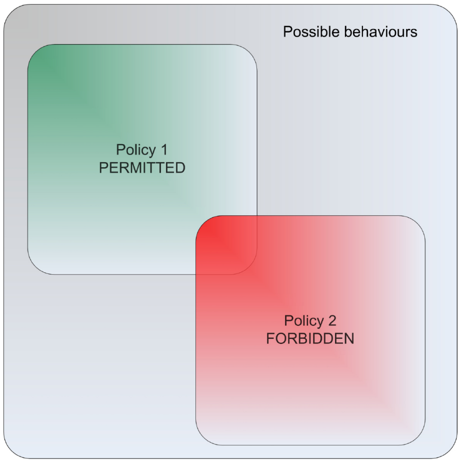


Fig. 2. Venn diagram of a mutually inconsistent pair of policies. The behaviors covered by both policies are the ones that are both permitted and forbidden.

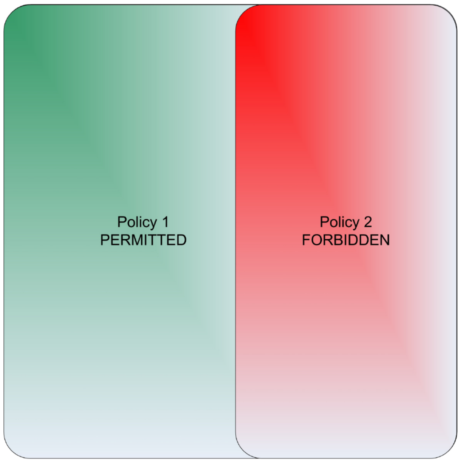


Fig. 3. Venn diagram of a mutually consistent pair of policies. The union of the two policies is an exhaustive policy, as there is no behavior that is neither permitted nor forbidden.

Next to consistency, we can also speak of exhaustiveness of policies, when there is no behavior that is neither permitted nor forbidden. Exhaustive policies cover all possible behaviors, and requiring a policy to be exhaustive makes sure that any possible behavior will be considered (Fig. 3).

Definition 5: A policy Θ is exhaustive iff for every behavior $T \in \mathcal{T}$, $\Theta \vdash P(T)$ or $\Theta \vdash \neg P(T)$.

In the area of security, problems with policies typically enable what we call attacks. Using the above definitions, we can define the notion of attack in terms of policies.

Definition 6: An attack on policy Θ_1 enabled by policy Θ_2 is a behavior that conforms to Θ_2, but violates Θ_1.

Typically, Θ_2 is an incorrect refinement of Θ_1. This refinement may have been explicitly designed as such, or it may be a policy implicitly defined by the technology and people in an organization.

[1] Although [10] use multisets of actions, we consider order important here, which will become clear when discussing the notion of local policies. We may wish to generalize sequences to partially ordered multisets in future work [26].

[2] See also [27] for an earlier example of using Venn diagrams in security assertions.

To be able to speak about the notion that a policy is a correct refinement of another policy, we need notions of soundness and completeness. Intuitively, soundness means that a refined policy does not break any requirements of the high-level policy, and completeness means that a refined policy covers everything that the high-level policy covers. These notions are defined purely on the syntactic level here, and thus do not have the usual interpretation in logic of soundness and completeness of theories with respect to models.

Definition 7: A policy Θ_2 is complete with respect to a policy Θ_1 iff for each formula ϕ such that $\Theta_1 \vdash \phi$, also $\Theta_2 \vdash \phi$.

This basic logical framework gives rise to some simple theorems, which we present here to provide a complete picture.

Theorem 1: If a policy Θ_2 is complete with respect to a policy Θ_1 that is inconsistent, then Θ_2 is also inconsistent.

Proof: If Θ_1 is inconsistent, then according to Definition 4 there is a formula ϕ such that $\Theta_1 \vdash \phi$ and $\Theta_1 \vdash \neg\phi$. As Θ_2 is complete with respect to Θ_1, this will mean according to Definition 6 that also (1) $\Theta_2 \vdash \phi$ and (2) $\Theta_2 \vdash \neg\phi$. Therefore Θ_2 is also inconsistent. ■

We call a policy sound with respect to another policy, if it does not violate the constraints of this other policy (allows a behavior forbidden by the other, or forbids a behavior allowed by the other).

Definition 8: A policy Θ_2 is sound with respect to a policy Θ_1 iff the policy $\Theta_1 \cup \Theta_2$ is consistent, i.e., if Θ_1 and Θ_2 are mutually consistent.

When refining policies, soundness thus expresses that the refinement does not go against the higher-level policy, whereas completeness expresses that everything of the higher-level policy is covered. Note that the soundness relation is symmetric. This may seem counterintuitive, but as long as policies are mutually consistent, they are sound refinements of each other in the sense that they do not contradict each other's requirements (although they are usually not complete). This signals something important. Although soundness seems to be an important property to express when refining policies, it is actually implied by the combination of consistency and completeness. Intuitively, this can be understood by the idea that if completeness holds for one policy with respect to another, then the permitted or forbidden conditions on behaviors must match for the behaviors that are covered by both policies, and conflicts between the policies can only occur if at least one of the policies is inconsistent. This is formalized in Theorem 2.

Theorem 2: If a policy Θ_2 is complete with respect to a consistent policy Θ_1, and Θ_2 is consistent itself, then Θ_2 is also sound with respect to Θ_1.

Proof: We prove the inverted statement: if a policy Θ_2 is not sound with respect to a consistent policy Θ_1, then either Θ_2 is not complete with respect to Θ_1, or Θ_2 is inconsistent. If a policy Θ_2 is unsound with respect to a policy Θ_1, then there exists ϕ such that $\Theta_1 \cup \Theta_2 \vdash \phi$ and $\Theta_1 \cup \Theta_2 \vdash \neg\phi$. If Θ_2 is not inconsistent itself, then there must be a ψ such that $\Theta_1 \vdash \psi$, $\Theta_2 \cup \psi$ is inconsistent, and $\Theta_2 \nvdash \psi$. Thus, there is a formula ψ that is derivable from Θ_1 but not from Θ_2. Therefore, Θ_2 is not complete with respect to Θ_1. ■

Definition 9: A policy Θ_2 is a proper refinement of a policy Θ_1, if Θ_2 is consistent, and Θ_2 is complete with respect to Θ_1.

It follows that a proper refinement is also sound. In principle, this definition allows us to judge whether a policy refinement is correct, using standard logic tools. However, such an analysis would often be unnecessarily complex, as many policies are stated in a limited number of formats.

IV. Types of Policies

In many cases, we do not need the full power of first-order logic to express policies. This also means that we can avoid problems of indecidability. The most limited policies are conjunctions of permitted or forbidden behaviors.

Definition 10: A simple policy is a set of sentences Θ of the form $P(T)$ or $\neg P(T)$.

A simple policy can be understood as assigning to each behavior a value: 1) don't care; 2) permitted; 3) forbidden; or 4) contradiction.

Many policies allowing certain behavior, however, require that a certain result can be achieved in relation to a business goal. Often, it is not of essential importance how this result is achieved. For example, there should be at least one possible way to change the configuration of the e-mail server. This means that security policies can forbid all but one of the concerned behaviors, as long as this one behavior remains possible. We can thus have a situation where out of a set of behaviors at least one should be possible.

Similarly, it would often be required that for an attack to be prevented, at least one of the constituting atomic behaviors (actions) should be disabled. Thus, a negative policy demanding exactly this would require at least one behavior in a set of behaviors to be impossible. We call such *at least one* policies extended policies.

Definition 11: An extended policy Θ is a set of sentences of the form $\phi_1 \vee \phi_2 \vee ... \vee \phi_n$, where each ϕ_i is of the form $P(T)$ or of the form $\neg P(T)$ with T a behavior.

Note that extended policies are only extended with respect to simple policies, not with respect to the general notion of policy defined in Definition 3. Extended policies are a subset of general policies, and simple policies are a subset of extended policies.

These types of policies are thus included in the general notion of policy, specified on traces. However, many real-life policies are not stated in terms of complete behaviors. Often, we see policies that are rather defined on:

1) the permissibility of actions given the preceding trace ("only people with a key should be able to open this room");
2) on the states of the system caused by the traces ("sales data should not end up outside of the organization").

This gives rise to two different kinds of policies that are not contained in the general notion. A local policy is a policy that specifies when a particular action can take place, based on the preceding sequence of actions. A state policy is a policy that specifies which states should be or should not be reachable.

Definition 12: A local policy is a theory Γ in first-order logic, with behaviors $T \in \mathcal{T}$ and actions $e \in \mathcal{E}$ being the terms and $L(\_, \_)$ a distinguished predicate over $\mathcal{T} \times \mathcal{E}$.

A local policy $L(T, e)$ thus expresses that action e is permitted following trace T.[3]

To specify policies on states, we need to augment the sequences of actions with an underlying system state representation, which we call a system model. Note that we only introduce the state representation at this point in this paper, and we still stick with our original interpretation of policies in terms of behaviors. However, in practice, policies are often specified on states, and to allow a translation from such policies to behaviors, we need to define their relation.

Definition 13: A system model $M = (\mathcal{S}, S_0, \mathcal{E}, \rightarrow)$ consists of a state space $\mathcal{S}$, an initial state S_0, a set of events $\mathcal{E}$, and a state transition function $\rightarrow : \mathcal{S} \times \mathcal{E} \rightarrow \mathcal{S}$.

We write $S_i \xrightarrow{e} S_j$ if there is a transition from state S_i to state S_j upon action e. We write $S_i \xrightarrow{T} S_k$ if there is a behavior $T = e_0 ... e_n$ that leads from state S_i to S_k by multiple transitions upon $e_0 ... e_n$, respectively.

Definition 14: A state policy is a theory Σ in first-order logic, with states $S \in \mathcal{S}$ being the terms and $G(\_)$ a distinguished predicate over $\mathcal{S}$.

In terms of our original definition of policy, a state policy would permit one of the behaviors leading to the specified state, or forbid all of the behaviors leading to the specified state. Thus, it describes reachability of the state in the system model.

Example 2: In the road apple example, there is a state policy forbidding all states in which sales data is outside the organization. In terms of behaviors, the policy means that such states should not be reachable. In the existing version of the organization, there are no local policies preventing behaviors that lead to such states. The challenge here is to define local policies that will do exactly this, for example, a local policy forbidding the connection of noncompany devices to company computers (potentially enforced by physical disabling). Even though this would not prevent the behavior of bringing devices to work (T), it would not allow the action of connecting them (e).

In the following sections, we will outline how the different types of policies can be compared against each other.

V. Checking Consistency and Completeness

As will be detailed in the related work (Section VIII), formalization of completeness of policies at different levels is one of our major contributions. In the following, we discuss how to assess consistency and completeness for the different types of policies that we distinguished. Our interpretation of policies in terms of behaviors is essential here, and enables a model-checking approach to finding mistakes.

Completeness of simple policies can be verified by checking if all permitted and forbidden behaviors of a high-level policy are also permitted and forbidden in the low-level policy. Don't cares in the high-level policy may be assigned any value in the low-level policy (although the value contradiction would make the low-level policy inconsistent, but not incomplete). Thus, checking consistency and completeness for simple policies is easy.

For extended policies, checking consistency requires the construction of a simple policy matching the criteria imposed by the extended policy. Verifying completeness of extended policies requires checking whether for each sentence in the high-level policy, there is a corresponding sentence in the low-level policy of which the disjunctive elements are a subset of those of the sentence in the high-level policy. Thus, the low-level policy should be more restrictive in terms of the set of behaviors of which at least one should be permitted or forbidden.

The most interesting case, however, is checking consistency and completeness of local policies against state policies. Local policies (usually delegated to agents within the system) enable or disable single actions, depending on the credentials, and thereby enable or disable particular traces on global system levels. A local policy can, for example, state that only persons with a key can enter a door. To check these for consistency and completeness against higher-level policies, possible sequences of actions need to be generated from the local policies to obtain global policies that can be compared against state policies, for example, stating that sales data should not leave the organization.

Definition 15: An implied global policy of a local policy Γ is a policy Θ such that for each behavior T and action e.

1) $\Theta \vdash P(\epsilon)$.
2) $\Theta \vdash P(Te)$ iff $\Theta \vdash P(T)$ and $\Gamma \vdash L(T, e)$.
3) $\Theta \vdash \neg P(Te)$ iff $\Theta \vdash \neg P(T)$ or $\Gamma \vdash \neg L(T, e)$.

Note that the predicate P of the implied global policy will be prefix closed (if $\Theta \vdash P(Te)$ then also $\Theta \vdash P(T)$). Intuitively, part 2 of the definition states that if an action is locally permitted following trace T, then either it should be globally permitted following trace T, or trace T should not be permitted at all. Thus, either the situation in which the action is allowed will not occur, or the action is enabled in that situation. Part 3 states that if an action is locally forbidden following trace T, then it should be globally forbidden following trace T. In the latter case, it does not matter whether T is globally permitted or forbidden, as Te should be globally forbidden in both cases.

Moreover, state policies also imply policies on sequences of actions, namely, those that lead to the specified states.

Definition 16: An implied global policy of a state policy Σ in system model $M = (\mathcal{S}, S_0, \mathcal{E}, \rightarrow)$ is a policy Θ such that for each behavior T:

1) if $\Sigma \vdash G(S)$, then there exists a behavior T such that $\Theta \vdash P(T)$ and $S_0 \xrightarrow{T} S$;
2) if $\Sigma \vdash \neg G(S)$, then for all behaviors T such that $S_0 \xrightarrow{T} S$, $\Theta \vdash \neg P(T)$.

Note that the former clause can also be represented as an extended policy (see Definition 11), where at least one of the behaviors leading to the state should be permitted.

Example 3: For the road apple example, the implied global policy of the state policy "sales data should not leave the

[3] In [25], this is referred to as action e being enabled for trace T.

organization" prohibits all behaviors that lead to such states. The local policy that forbids connection of external devices to company computers implies a global policy that prohibits all behaviors that contain such actions. In this local policy, there is no constraint on the preceding trace, such as when a key is required to open a door.

To test completeness of local policies with respect to state policies, the policies need to be translated to standard policies in terms of behaviors. To do this, we need to determine whether composite behaviors are permitted or forbidden based on the regulation of individual actions by the low-level policies. The local (low-level) policies are used to generate the traces that they allow, and the states that are reachable. For example, if a laptop is located in a room, and I cannot access the room without the key, then the sequence of accessing the room and removing the laptop is not permitted. However, the sequence of asking someone the key, accessing the room, and removing the laptop might be possible (permitted). If the initial policies are only defined locally, this will not lead to inconsistencies. Next, the permitted and forbidden traces and states are tested against the state (high-level) policies, which may or may not prove completeness. The policies are complete if all permitted states are reachable, and all forbidden states are unreachable. If a violation of the state policy forbidding certain states can occur, this corresponds to a behavior that satisfies the local policies, but violates the state policies, proving incompleteness, and suggesting an attack.

The question is which methods are most appropriate for such an analysis. When all implied global policies would be determined, consistency and completeness could be assessed with standard logic tools. However, the explicit generation of all behaviors that would be needed to specify the implied global policies is very inefficient, as the possibilities for interaction typically grow exponentially with the number of entities in a system. Even in relatively simple systems, the size of the behavior space would therefore become prohibitively large, and cyclic behaviors would require methods to cut infinite traces.

Therefore, a first step is a simplification of the trace representation. Then, it would still be possible to use existing tool support, but we concluded earlier from a small experiment that smarter solutions are needed for specific cases [4, ch. 4]. In the following, we present a solution for the most typical case, i.e., comparing local policies against state policies (only people with a key can open this door versus sales data should not end up outside the organization). Although this is not a fully general solution, it covers a typical security scenario: behaviors that lead to a state in which security is violated should not be possible. Therefore, this is a case that is of particular interest to the security community.

Furthermore, the proposed approach closely links up with existing methods in the field, such as attack trees. As the translation of local policies to global ones will generate particular enabled traces, notably ones that conflict with existing global policies, the procedure is exactly the same as procedures that are used for generating attack scenarios from system models with local policies. The only difference here is that we make the notions of global and local policies explicit, providing a sound conceptual and formal framework for existing approaches to attack generation. Thereby, we explain how existing methods of analysis for finding attacks can be expressed in terms of policy alignment.

VI. Attack Generation

A. *Definitions*

System models for attack generation can be understood as completeness checkers for policies. They will compare local decisions (local policies) against global requirements (state policies). Typically, a system model underlying attack generation methods is specified as a graph and the local policies are assigned to nodes in the graph [12]–[14]. The edges, which represent access relations between entities, then represent the system state. For example, an edge between a person and a room would indicate that the person has access to the room. In the case the local policies assigned to the nodes are dynamic, they are also part of the state. Such system models thus allow the representation of both local policies assigned to agents, as well as state policies on the state of the system as a whole. Based on the model and the above definitions, they can be translated to regular policies.

To enable a completeness analysis, the system infrastructure is represented with the imposed (local) policies. As said, these are usually formulated in terms of credentials, locations, and identities required for an action e. Possession of the required items can be derived from the preceding trace. For example, a door connects two rooms (infrastructure), and a key is needed to open the door (policy). Whether the agent will have the key will depend on the preceding trace. The state of the infrastructure can thus change over time, for example, if agents obtain keys. To connect the state policies and the local policies, we specify system model states in terms of attributes [28], i.e., edges representing access relations between nodes in the graph. State policies can then refer to the attributes of the states considered (e.g., sales data not being outside of the organization), and local policies can refer to the attributes as a means to represent the preceding trace (e.g., a person being in possession of a key). An action now consists of the satisfaction of an attribute, or the addition of an edge to the graph.

The system's local policies are represented as preconditions of attributes in terms of other attributes. To enable the connection of different attributes in a trace or behavior, the attributes are annotated with the preconditions that can lead to the attribute being satisfied. For example, the attribute representing that I have access to a room has the preconditions that I have access to the hall, and that I have access to the key.

Definition 17: An attribute a is a pair (name, precondition). The precondition is specified as a set of sets of other attributes, corresponding to a disjunctive normal form (disjunction of conjunctions). A state is expressed as a predicate S on attributes. An action consists of the transition of the predicate value of one attribute from $\perp$ to $\top$. If the precondition is $\top$, the attribute is satisfied in the initial state.

The set of attributes for a particular system is denoted as $\mathcal{A}$. We often write only the name to refer to an attribute, omitting the precondition for brevity.

The state representation thus consists of a representation of whether attributes are satisfied (whether edges exist in the graph). State policies can often be understood as predicates over attributes, either: 1) some attribute should be satisfiable (being part of an essential business process), or 2) it should be unsatisfiable (being a security threat). High-level policies then state that either: 1) at least one behavior leading to the satisfaction of the attribute should be permitted, or 2) all behaviors leading to the satisfaction of the attribute should not be permitted.

As in [28], it is assumed that satisfied attributes remain satisfied forever. This is called the monotonicity assumption, and it is a useful heuristic to prevent state explosion and infinite loops. Only in cases where an agent has to go back to a previous location, this assumption would not find the real traces, and actions for going back would have to be added by an additional procedure. Monotonicity leads to an overapproximation, finding traces that cannot occur in practice, when an agent is not able to go back. In such cases, the procedure will conclude that certain traces are allowed by the low-level policies, whereas they are actually not. If such a trace is forbidden by the high-level policy, the procedure will conclude that the low-level policies are not complete with respect to the high-level policy. Conversely, when a certain trace should be possible according to the high-level policies, it may be found that the low-level policies are complete (they seem to permit the behavior), whereas they are actually not. However, such cases are very rare in practice. They would be relevant, though, when focusing on systems that attempt to prevent an intruder from escaping with his catch. Here, we are primarily interested in testing whether gaining access is possible.

Attributes thus express properties of the world that may become satisfied over time. To make attributes workable, they have to be generalized beyond individual objects. For example, the attribute $(Steve\_in\_room, \{\{Steve\_in\_hall, key\_in\_hall\}\})$ states that Steve can enter the room when he is in the hall and the key is in the hall. (Remember that the precondition is a set of sets of attributes corresponding to a disjunctive normal form.) It should be generalized to express $(in\_room(x), \{\{in\_hall(x), in\_hall(key)\}\})$, with x a variable. This is then an attribute template that covers many individual instances.

When expressed in logic, this can also be written as $\forall x : in\_hall(x) \wedge in\_hall(key) \rightarrow in\_room(x)$. In this case, state transitions are replaced by derivation steps. We can even generalize one level higher, and then obtain $\forall x : in(x, hall) \wedge in(key, hall) \rightarrow in(x, room)$.

If we have the "in" relation relating entities to groups of entities as the only relation, we can represent the attributes by a hypergraph, as in the ANKH system model [14]. Attributes then represent which new group memberships are possible based on which existing group memberships. The ANKH model, in addition, constrains the policies by requiring that in order for a new group membership to be possible based on an existing group membership, there must exist an entity that is a member of both groups already, and this entity must explicitly allow the new membership based on specified preconditions. In this way, different system models constrain the attributes (and thus the associated preconditions) in different ways *a priori*, typically based on some notion of proximity of entities (actions cannot take place from a distance).

Attributes give rise to another type of policy, specifying whether attributes are permitted or not.

Definition 18: An attribute policy is a theory Φ in first-order logic, with attributes $a \in \mathcal{A}$ being the predicates.

Typically, state policies can be expressed more compactly as attribute policies: if sales data should not leave the organization, we can prohibit all states in which the sales data is outside, but we can write an equivalent attribute policy that simply prohibits the attribute.

B. Method

With respect to the goal of describing attack generation as policy completeness checking, we now know how state representations with attributes in system models are related to policies in our policy alignment framework: local policies describe how attributes can change (which actions are possible), and attribute policies (state policies) describe which attributes should or should not occur (which states are required or forbidden).

By understanding actions as satisfaction of attributes, we can now define a procedure for testing completeness of local policies against state policies with system models (Algorithm 1). Informally, the algorithm is as follows.

1) Assume that all attributes with precondition $\top$ are satisfied.
2) Check which attributes now have their precondition satisfied, and mark these as satisfied.
3) Repeat until no more attributes can be satisfied.
4) Check if the attribute policy is violated by the final set of satisfied attributes.
5) If so, trace back attributes to its original preconditions, and output possible attacks in terms of sequential satisfaction of attributes.

When an attribute policy is input to the model that prohibits certain attributes, the analysis will aim at finding a behavior that is allowed by the local policies, but still leads to satisfaction of the particular attribute (i.e., violates completeness). In this case, we can easily check the completeness of local policies with respect to such a policy by the above procedure. We only have to judge whether the corresponding attribute is satisfied after execution of the method. If we wish to know what behaviors can lead to the satisfaction of the attribute, we can backtrack the analysis following the preconditions of the attributes, up to the point where all remaining attributes have precondition $\top$ [4, p. 78]. It would be interesting to investigate how such an analysis might work for other than simple attribute policies.

By following the traces back from prohibited states toward the initial state, one can build an attack tree [10], [29], in which all the possibilities for violating the associated state policy are visualized. In practice, because attributes can occur in preconditions multiple times, the attack tree may not actually be a tree in the mathematical sense, but rather a graph.

However, as the notion of attack graph has a different meaning in security analysis, we call the resulting structures attack trees anyway.

Algorithm 1 Attack generation or completeness checking

```
Inputs:
  • system model with attributes $\mathcal{A}$;
  • simple attribute policy $\Phi$.
Outputs:
  • violated policy sentences of the attribute policy, and
    corresponding possible attacks.
Local variables:
  • set of satisfied attributes $A_\top$;
  • set of newly satisfied attributes $A_{new}$;
  • set of attributes to be checked plus their relevant
    precondition sentence $A_{ch}$;
  • iteration number $i$;
  • tracking table $M$ containing entries of the form (attribute name,
    iteration number, satisfied conjunctive clause);
  • conflicting sentences in policy $C$;
  • possible attacks $T$;
  • result, consisting of tuples (conflicting sentence, possible attacks) $R$.
Algorithm (pseudocode):
  $M \leftarrow \emptyset$
  $A_\top \leftarrow \emptyset$
  $A_{new} \leftarrow \{(n, \top) \in \mathcal{A}\}$
  $i \leftarrow 0$
  repeat
    $A_\top \leftarrow A_\top \cup A_{new}$
    $A_{ch} \leftarrow \{(n, P, p) \mid (n, P) \in \mathcal{A} \setminus A_\top \wedge p \in P \wedge p \cap A_{new} \neq \emptyset\}$
    $A_{new} \leftarrow \emptyset$
    for $(n, P, p) \in A_{ch}$ do
      if $p \subseteq A_\top$ then
        $M \leftarrow M \cup \{(n, i, p)\}$
        $A_{new} \leftarrow A_{new} \cup \{(n, P)\}$
      end if
    end for
    $i \leftarrow i + 1$
  until $A_{new} = \emptyset$
  $A_\top \leftarrow A_\top \cup \{\neg a \mid a \in \mathcal{A} \setminus A_\top\}$
  $C \leftarrow \{s \in \Phi \mid A_\top \cup \{s\} \vdash \bot\}$
  $R \leftarrow \emptyset$
  for $c \in C$ do
    $T \leftarrow \{(n, i, p) \in M \mid (n, i, p)$ is a step contributing to the
      final contradiction of $c\}$[4]
    $R \leftarrow R \cup \{(c, T)\}$
  end for
  return R
```

Example 4: The state policy in our running example (the road apple attack) forbids all sequences of actions that result in the sales data being outside of the organization. This can be expressed as an attribute policy (with states interpreted as predicates over attributes)

$$\Phi = \{\neg in(salesdata, outside)\} \tag{1}$$

which is equivalent to a state policy forbidding all states where this attribute is satisfied

$$\begin{aligned}\Sigma = \{\forall S \in \mathcal{S} :\\ S(in(salesdata, outside)) = \top \rightarrow \neg G(S)\}\end{aligned} \tag{2}$$

which is again equivalent to a policy forbidding all behaviors leading to such states

$$\begin{aligned}\Theta = \{\forall T \in \mathcal{T}, S \in \mathcal{S} :\\ S(in(salesdata, outside)) = \top \wedge (S_0 \xrightarrow{T} S)\\ \rightarrow \neg P(T)\}.\end{aligned} \tag{3}$$

If all of the actions constituting one such behavior are possible (permitted by the local policies), they constitute an attack on the high-level policy. Using graph-based system model analysis, such attacks can be determined by Algorithm 1. In the road apple example, the road apple attack is enabled by the local policies, and will therefore be output by the algorithm. Depending on the other entities represented in the model, other attack scenarios might be possible as well. The high-level and low-level policies are thus not properly aligned. To achieve alignment, at least one of the actions constituting the road-apple attack should be disabled by a local policy. Thereby, other attacks containing this action are also disabled. The analysis can be rerun with simulated countermeasures to determine the overall effect of such measures on possible attacks.

In Fig. 4, the possible behaviors violating the high-level policy are represented in an attack tree for the road apple example [14].

Similarly, it may be checked whether it would be possible to send aggregate sales data to the tax office. Even with security policies in place, high-level policies may state that there should still be a way to do this. In this case, the analysis amounts to checking whether something is possible rather than impossible. In this case, the high-level policy actually states that at least one behavior leading to the target situation should be possible, i.e., it is an extended policy.

In case we wish to disable the road apple attack, at least one of the constituting actions should be disabled, giving a negative extended policy. When composing such policies (required to disable attacks), the question of whether the policies are still consistent becomes relevant.

C. Scalability

We have run experiments by comparing local policies against attribute policies using the Portunes system model [4]. This analysis is aimed at finding a violation of a state or attribute policy that is permitted by the local policies. With standard model checking tools, experimental results show a $O(N^6)$ complexity. N represents the number of nodes in the model, and we assume that the number of local policies is

[4] For additional details, see [4, Algorithm 2, p. 78]. For sentences consisting of nonnegated attributes (stating that a certain attribute should be satisfiable), it is not possible to generate scenarios explaining why the policy is not satisfied (there is no concrete counterexample). It might be feasible to say something about the closest possibility of satisfying the attribute, but that would be future work, which the general structure of the algorithm allows.

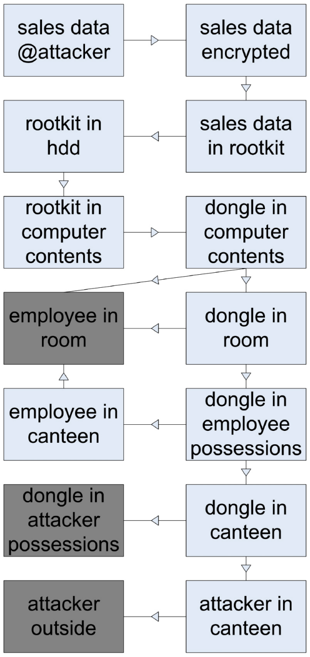


Fig. 4. Attack tree for the road apple example (adapted from [14]). The attack tree represents the possible behaviors that violate the policy "no sales data outside organization." The grey boxes are attributes with precondition $\top$ (initially satisfied attributes). Arrows point to attributes required by the precondition of their originating attribute.

in the same order. With dedicated algorithms, with theoretical worst-case complexity of $O(N^4)$, the experiments give $O(N^{3.3})$ and $O(N^{1.7})$ for constructed examples with expected bad and good scalability behaviors, respectively.

In these algorithms, the monotonicity assumption [28] simplifies the calculations by requiring that edges (attributes) can only be added to the graph of the system model, not removed. This is adequate for most practical cases. Most of the complexity lies in calculating all satisfiable attributes. When this has been done, finding out which local policies are responsible for the violation of a different state or attribute policy is relatively cheap ($O(N^2)$). For details, see [4, ch. 4].

VII. Applications

In summary, the framework of policy alignment provides a formal foundation for the analyses for finding attack scenarios in socio-technical systems. The present formalization provides a theoretical foundation in terms of:

1) explicit definition of policies in terms of behaviors;
2) description of high-level and low-level policies in terms of permitted and forbidden behaviors, thereby explicating the link between high- and low-level policies;
3) understanding of attack generation as generation of behaviors that violate the high-level policies (typically state or attribute policies).

Based on the above outline, different applications of our formalization of policy alignment are possible, or will become possible through further efforts.

A. Predicting Attacks by Misalignment Analysis

As outlined above, the completeness analysis of local policies against global policies can be used for predicting possible attacks in socio-technical systems. Although such methods were proposed before, we are the first to formalize this idea in terms of incompleteness of policies. In addition to the basic analysis, observed traces can be used to test whether the system conforms to the policies (see [30]).

B. Attack Trees as Policies

Attack trees [29] are trees that show how an attacker can reach a certain goal (root node). The tree splits when an attacker has to execute multiple actions (AND node) or can choose between actions (OR node) to achieve a goal. Mauw and Oostdijk [10] provided a formal semantics for attack trees. The semantics of an attack tree is a multiset of actions, namely, those that lead to the target situation of the attack tree.

In our work on policy alignment, we are interested in policies that separate between permitted/forbidden or possible/impossible behaviors. An attack tree can, therefore, also be seen as a policy that allows exactly the behaviors of its semantics. As a policy, it may be conflicting with a policy that forbids such behaviors. In particular, higher-level policies will typically prohibit behaviors that lead to a situation represented as the goal of an attack. In this case, the behaviors described by the attack tree will conflict with the higher-level policy.

Conversely, an attack tree may also be seen as a policy forbidding the behaviors that constitute the tree, i.e., all the behaviors that achieve the goal of the attack. The attack tree then becomes a specification of defensive measures needed to prevent the attack. Such a policy will be the union of a set of extended policies, namely, for each behavior reaching the goal, at least one of the constituting actions should be forbidden.

C. Representing Multilevel Authorizations

Normally, the formal study of authorizations is limited to authorizations on one level: persons are mapped to roles, and roles are mapped to access to objects (see [15]). Even when refinement is discussed, as in [31], this refinement only considers single actions and the associated authorizations. However, in organizations one typically wants certain persons or roles to achieve certain outcomes, but at the same time one wants to prevent other results. Thus, when someone is authorized to send aggregate sales data to shareholders, this person should be permitted to execute all actions constituting one of the possibilities to achieve this goal. In other words, there is an alignment question here in terms of how to define the low-level authorizations such that this high-level authorization is effectuated. Moreover, one will often want this to happen without giving low-level authorizations that can lead to undesirable outcomes (insider attacks). This provides a detailed account of how to implement least privilege by means of policy alignment with multiple authorization levels.

D. Quantification

In addition to describing policies in terms of permitted and forbidden, it would be interesting to look at quantitative values. These values would then represent the difficulty or cost of a behavior [32]. Policies could specify a maximum or minimum difficulty for sets of behaviors, where minimum difficulty would correspond to a security requirement (things that should not happen should be difficult), and maximum difficulty would correspond to a usability requirement (things that should happen should be easy). Fuzzy logic could be used to support such policies.

Consistency and completeness will have different meanings for quantified policies. Rather than binary values, they will now also be quantitative values indicating goodness of policies. This will also require different definitions of consistency and completeness.

These quantification efforts correspond to the labeling of attack trees with different types of values (likelihood, effort, cost, reward, etc.). However, they are now part of a model of the socio-technical system infrastructure rather than of a pre-defined attack tree. This means that attack trees will first have to be generated from the system model to determine the total difficulty of particular attacks. Heuristics may need to be applied to keep the model checking manageable.

E. Policy Design

Ultimately, the goal of this paper is to allow system designers to specify low-level policies based on high-level policies defined on the management level in organizations. A full-fledged method for achieving this goal will require further research in the areas outlined above.

VIII. Related Work

A. Policy Alignment

Abrams and Bailey [20] discussed the refinement of security policies across different levels of abstraction, where lower-level policies are implementations of higher-level policies. They discussed consistency and conformance of policies between levels. They did not formalize these relations, and neither did they discuss the possibility that not all behaviors will be categorized as permitted or forbidden at higher levels of abstraction. Nunes Leal Franqueira and Van Eck [33] discussed alignment of policies between different domains (access control, network layout, and physical infrastructure) based on the formalism of law-governed interactions. They only focused on expressing policies from the different domains in a single language, not on refinement and completeness of policies.

B. Security Logics

Cholvy, Cuppens, and others [6], [7] focused on consistency of security policies and the merging of policies based on deontic security logics, using an operator for obligation (where forbidding is expressed as obligation not to do something, and permission is expressed as not being obliged not to do something). They focused on logical access, and discussed whether it is possible for situations to occur in which there is a conflict. For example, if a user cannot downgrade a file, but a system security officer (SSO) can, and it is at the same time specified that a SSO is also a user, then if there exists an agent with the role of SSO, and a file, then downgrading the file by the SSO is both permitted and forbidden.

We are also interested in conflict situations, but: 1) we focus on socio-technical systems, with multistep attacks, including physical access and organizational policies, and 2) we relate our work to model checking instead of theorem proving. In addition, we do not discuss obligatory actions, as we are primarily interested in the possibility or impossibility of attacks through permitted and forbidden actions. We thus do not need to use deontic logic, as we only consider whether actions are permitted or forbidden at a specific level of abstraction. At a high level, this has a normative meaning ("sales data should not (cannot) leave the organization"); at a low level, this has a descriptive meaning ("this door can only be opened with the key"). This difference does not impact the analysis, as we only focus on the alignment of the policies between the different levels of abstraction.

C. Security Policy Refinement

Several papers discussed policy refinement for specific scenarios. For example, Craven *et al.* [31] discussed policy refinement in a database setting, and Laborde, Barrère, and Benzekri focused on policy refinement in networks [34]. Bonatti, De Capitani di Vimercati, and Samarati [5] focused on the composition of multiple policies, where policies may be underspecified. These approaches achieve major improvements in the flexibility of reasoning on security policies, by providing representations of policies at different levels of abstraction that are specific to the context considered. Here, however, we are interested in multistep scenarios in socio-technical systems, and because of the complexity and the many different types of actors involved, we need a more general formalization. In particular, previous work discusses policies that are already formulated in terms of subjects, objects, and individual actions, but refines these following refinement of the subjects, objects and actions. For policy alignment and refinement to work in socio-technical system scenarios, we are interested in how to refine policies that can be expressed in terms of more complex behaviors (i.e., sequences of actions).

D. Consistency and Completeness

Checking consistency of security policies has been discussed in the literature from an intensional or theorem proving point of view [6], [7]. From a model-checking point of view, consistency of policies depends on the space of behaviors. Therefore, the possible behaviors first need to be generated to determine whether the policies imposed on the behaviors are consistent. In practice, conflicts could, for example, occur when particular policies apply in emergency situations, such as doors that are automatically unlocked, whereas security policies would require the doors to be closed (i.e., allow no behaviors that involve opening the doors).

Checking completeness of security policies is less well studied, because the notion of policies at different levels of abstraction has not been taken into account. Where completeness is mentioned, e.g., in [15] and [35], it refers to what we have called exhaustiveness. Formally, discussing completeness of policies at different levels is therefore a major contribution.

E. System Models

System models [12]–[14], [36] are representations of an organization's technical and social infrastructure, aimed at finding security vulnerabilities in the infrastructure. Attacks (or attack trees) can be automatically generated from such models. The models check a high-level policy (e.g., "sales data should not leave the organization") against low-level policies (e.g., "this door can only be opened with a special key"). Intuitively, this can be understood as a form of policy alignment, and one of our important contributions is a more precise definition of this relation.

As we are interested in possible policy violations, or (in)completeness of refined policies with respect to high-level ones, we focus on an extensional interpretation of policies here, i.e., in terms of the set of behaviors that they permit or forbid. The extensional interpretation also makes it possible to visualize the policies in Venn diagrams by showing policies in terms of permitted and forbidden subsets of behaviors. It also enables model checking for consistency and completeness by systematically exploring the space of behaviors.

IX. Conclusion

In this paper, we formalized the notion of security policy alignment. Policy alignment has been known as an approach for assessment of organizational security policies, but a formal foundation was lacking. This meant that the relation between the areas of (informal) policy alignment, security logics, and system models remained implicit. Our formalization provides a formal foundation for model-checking approaches to finding security weaknesses in complex socio-technical systems, based on the up-to-now-informal notion of security policy alignment.

Our formalization of security policies is based on theories in first-order logic, with a permission predicate over behaviors. Security policies can then be checked for consistency and completeness. We showed that soundness can be expressed as a combination of these. Completeness of local policies delegated to agents can be checked with system models, by comparing the traces that they allow against global policies stated in terms of states or attributes. This provides a clear foundation for the relation between system models and security policies. To allow other than black-or-white policies, which is typical when policies are delegated to humans, we sketched possibilities to transform the definitions and checks to a quantitative setting.

The model could further be extended with policies representing obligation (see [9]). This is especially relevant for local policies, as these denote which next actions are permitted given the preceding trace (expressed in the attributes of the state). Based on the preceding trace, it could also be specified that a particular action is compulsory. In the behavior, such an action should then always be executed first, before any other actions can take place. This would then, in turn, imply a global policy, by preventing certain behaviors (namely those that do have different actions before the compulsory one). However, such an analysis is not completely trivial within the proposed framework, especially in relation to the monotonicity assumption that is introduced to keep the analysis scalable. If attributes can only be added, but not removed, there would not be any reason to execute actions in a particular order, as the preconditions will never become false again after they were true once. For obligation to be meaningful, the obligatory action would have to disable others (such as when locking a door), and therefore requires lifting of the monotonicity assumption. Whether this keeps the analysis scalable remains to be seen. Still, we showed that for many security problems, focusing only on what is possible or permitted already provides valuable results.

Another topic for future work is the integration of the present formalism of policy alignment with our previous work on system refinement [23]. It would then become possible to analyze whether an attack that would be possible or impossible in a system would still be possible or impossible in a refinement of that system.

We also hope to further develop quantitative models for security analysis, based on the present formalization. Such models would be able to assist companies in estimating the likelihood, difficulty, and damage of attacks, as well as the effectiveness of countermeasures in reducing the values of these variables.

References

[1] M. Corpuz and P. Barnes, "Integrating information security policy management with corporate risk management for strategic alignment," in *Proc. 14th World Multi-Conf. Systemics Cybern. Informatics*, 2010, pp. 337–342.

[2] A. Creery and E. Byres, "Industrial cybersecurity for a power system and SCADA networks: Be secure," *IEEE Ind. Appl. Mag.*, vol. 13, no. 4, pp. 49–55, Jul. 2007.

[3] N. F. Doherty and H. Fulford, "Aligning the information security policy with the strategic information systems plan," *Comput. Security*, vol. 25, no. 1, pp. 55–63, Feb. 2006.

[4] T. Dimkov, "Alignment of organizational security policies: Theory and practice," Ph.D. dissertation, Faculty Electr. Eng., Math. Comput. Sci., Univ. Twente, Enschede, The Netherlands, Feb. 2012.

[5] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati, "An algebra for composing access control policies," *ACM Trans. Inform. Syst. Security*, vol. 5, no. 1, pp. 1–35, Feb. 2002.

[6] L. Cholvy and F. Cuppens, "Analyzing consistency of security policies," in *Proc. IEEE Symp. Security Privacy*, May 1997, pp. 103–112.

[7] F. Cuppens, L. Cholvy, C. Saurel, and J. Carrere, "Merging security policies: Analysis of a practical example," in *Proc. 11th IEEE Comput. Security Found. Workshop*, Jun. 1998, pp. 123–136.

[8] H. Hamed and E. Al-Shaer, "Taxonomy of conflicts in network security policies," *IEEE Commun. Mag.*, vol. 44, no. 3, pp. 134–141, Mar. 2006.

[9] B. Solhaug and K. Stølen, "Preservation of policy adherence under refinement," *Int. J. Software Informatics*, vol. 5, nos. 1–2, pp. 139–157, 2011.

[10] S. Mauw and M. Oostdijk, "Foundations of attack trees," in *Proc. 8th Annu. Int. Conf. ICISC'05*, LNCS 3935. 2006, pp. 186–198.

[11] B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer, "Foundations of attack–defense trees," in *Proc. 7th Int. Workshop FAST'10*, LNCS 6561. 2011, pp. 80–95.

[12] C. W. Probst and R. R. Hansen, "An extensible analysable system model," *Inform. Security Tech. Rep.*, vol. 13, no. 4, pp. 235–246, 2008.

[13] T. Dimkov, W. Pieters, and P. Hartel, "Portunes: Representing attack scenarios spanning through the physical, digital and social domain," in *Proc. Joint Workshop ARSPA-WITS Revised Selected Papers*, LNCS 6186. Mar. 2010, pp. 112–129.

[14] W. Pieters, "Representing humans in system security models: An actor-network approach," *J. Wireless Mobile Netw. Ubiquitous Comput. Dependable Appl.*, vol. 2, no. 1, pp. 75–92, 2011.

[15] S. Jajodia, P. Samarati, and V. Subrahmanian, "A logical language for expressing authorizations," in *Proc. IEEE Symp. Security Privacy*, May 1997, pp. 31–42.

[16] R. Baskerville and M. Siponen, "An information security meta-policy for emergent organizations," *Logistics Inform. Manage.*, vol. 15, nos. 5–6, pp. 337–346, 2002.

[17] J. Rees, S. Bandyopadhyay, and E. H. Spafford, "PFIRES, a policy framework for information security," *Commun. ACM*, vol. 46, no. 7, pp. 101–106, Jul. 2003.

[18] C. W. Probst, R. R. Hansen, and F. Nielson, "Where can an insider attack?" in *Formal Aspects in Security and Trust* (LNCS, vol. 4691). Berlin, Germany: Springer, 2007, pp. 127–142.

[19] I. M. Olson and M. D. Abrams, *Infomation Security Policy*. New York: IEEE Computer Society Press, 1995, pp. 160–169.

[20] M. Abrams and D. Bailey, "Abstraction and refinement of layered security policy," in *Information Security: An Integrated Collection of Essays*, M. Abrams, S. Jajodia, and H. Podell, Eds. New York: IEEE Computer Society Press, 1995, pp. 126–136.

[21] C.-M. Karat, C. Brodie, and J. Karat, "Usable privacy and security for personal information management," *Commun. ACM*, vol. 49, no. 1, p. 56, Jan. 2006.

[22] D. Sannella and A. Tarlecki, *Foundations of Algebraic Specification and Formal Software Development* (EATCS Monographs on Theoretical Computer Science). Berlin, Germany: Springer, 2012.

[23] D. Pavlovic and D. Smith, "Software development by refinement," in *Formal Methods at the Crossroads* (LNCS, vol. 2757), B. K. Aichernig and T. Maibaum, Eds. Berlin, Germany: Springer, 2003.

[24] S. Stasiukonis. (2006). *Social Engineering the USB Way* [Online]. Available: http://www.darkreading.com/document.asp?doc_id=95556

[25] J. Feigenbaum, A. Jaggard, and R. Wright, "Toward a formal model of accountability," in *Proc. New Security Paradigms Workshop*, 2011, pp. 45–56.

[26] V. Pratt, "Modeling concurrency with partial orders," *Int. J. Parallel Programming*, vol. 15, no. 1, pp. 33–71, Feb. 1986.

[27] J. Michael, E. Sibley, and D. Littmann, "Integration of formal and heuristic reasoning as a basis for testing and debugging computer security policy," in *Proc. NSPW*, 1993, pp. 69–75.

[28] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proc. 9th ACM Conf. Comput. Commun. Security*, 2002, pp. 217–224.

[29] B. Schneier, "Attack trees: Modeling security threats," *Dr. Dobb's J.*, vol. 24, no. 12, pp. 21–29, Dec. 1999.

[30] C. W. Probst and R. R. Hansen, "Analysing access control specifications," in *Proc. 4th IEEE Int. Workshop SADFE*, May 2009, pp. 22–33.

[31] R. Craven, J. Lobo, E. Lupu, A. Russo, and M. Sloman, "Security policy refinement using data integration: A position paper," in *Proc. 2nd ACM Workshop Assurable Usable Security Configuration*, 2009, pp. 25–28.

[32] G. Wyss, J. Clem, J. Darby, K. Dunphy-Guzman, J. Hinton, and K. Mitchiner, "Risk-based cost-benefit analysis for security assessment problems," in *Proc. IEEE Int. Carnahan Conf. Security Technol.*, Oct 2010, pp. 286–295.

[33] V. Nunes Leal Franqueira and P. A. T. van Eck, "Toward alignment of architectural domains in security policy specifications," in *Proc. 8th Int. Symp. Syst. Inform. Security*, Nov. 2006 [Online]. Available: http://eprints.eemcs.utwente.nl/7542/

[34] R. Laborde, F. Barrère, and A. Benzekri, "Network security policy refinement process: Expression and analysis," *J. High Speed Netw.*, vol. 15, no. 3, pp. 247–260, 2006.

[35] C. Bidan and V. Issarny, "Dealing with multi-policy security in large open distributed systems," in *Proc. 5th Eur. Symp. Res. Comput. Security*, LNCS 1485. 1998, pp. 51–66.

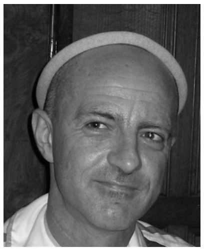
[36] D. Pavlovic and C. Meadows, "Actor network procedures," in *Proc. Int. Conf. Distributed Comput. Internet Technol.*, LNCS 7154. 2012, pp. 7–26.

Wolter Pieters received Masters degrees in computer science and philosophy of technology from the University of Twente, Enschede, The Netherlands, and the Ph.D. degree in risks and security of electronic voting systems from Radboud University Nijmegen, Nijmegen, The Netherlands.

He is currently a Post-Doctoral Researcher with the Energy and Industry Section, Faculty of Technology, Policy, and Management, Delft University of Technology, Delft, The Netherlands, addressing risks and security in smart energy systems. He was an Advisor to the Dutch Ministry of the Interior on Information Security, and then joined the University of Twente as a Researcher on sociotechnical cyber risks. He has published widely in the topics of cloud security, attack modeling, and cyber security ethics.

Trajce Dimkov received a degree in electrical engineering from the Faculty for Electrical Engineering and Information Technologies, SS Cyrili and Methodius University, Macedonia, and the Ph.D. degree from the Distributed and Embedded Security Group, University of Twente, Enschede, The Netherlands, with a thesis on "alignment of organizational security policies."

After the Ph.D. degree, he has been with the Security and Privacy Group, Deloitte, The Netherlands. His current research interests include formal modeling and penetration testing experiments, including social engineering.



Dusko Pavlovic (M'96) grew up in Sarajevo, Bosnia and Herzegovina. He studied mathematics and computer science in Utrecht, The Netherlands, where he received the M.Sc. and Ph.D. degrees.

He has taught at McGill University, Montreal, QC, Canada, the Imperial College, London, U.K., the University of Sussex, Sussex, U.K., and the University of Oxford, Oxford, U.K. He worked on software and security research for ten years in Palo Alto, CA, before rejoining academia as a Professor of information security at the Information Security Group, Royal Holloway, University of London, London, where he recently founded the Adaptive Security and Economics Laboratory. He is the part-time Chair in security with the Distributed and Embedded Security Group, Faculty of Electrical Engineering, Mathematics, and Computer Science, University of Twente, Enschede, The Netherlands. His current research interests include mathematics of security and network computation.