Security Analysis of Socio-Technical Physical Systems

Gabriele Lenzini[a], Sjouke Mauw[a,b], Samir Ouchani[a]

[a] *Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg*
[b] *Faculty of Communication and Computer Science, University of Luxembourg, Luxembourg*

Abstract

Recent initiatives that evaluate the security of physical systems with objects as assets and people as agents –here called *Socio-Technical Physical Systems*– have limitations: their agent behaviour is too simple, they just estimate feasibility and not the likelihood of attacks, or they do estimate likelihood but on explicitly provided attacks only. We propose a model that can detect and quantify attacks. It has a rich set of agent actions with associated probability and cost. We also propose a threat model, an intruder that can misbehave and that competes with honest agents. The intruder's actions have an associated cost and are constrained to be realistic. We map our model to a probabilistic symbolic model checker and we express templates of security properties in the Probabilistic Computation Tree Logic, thus supporting automatic analysis of security properties. A use case shows the effectiveness of our approach.

Keywords: Socio-Technical Systems, Security Assessment, Probabilistic Verification, Security Model.

1. Introduction

Surely anyone raises an eyebrow when reading about simple yet canny social engineering attacks like the one first reported in [1] and recently recalled by Schneier in his blog [2]. Without having the key and without breaching the door, an intruder stole a confidential document from a supposed-to-be locked room. This was possible because someone executed what the intruder's note requested: 'please leave the door unlocked'. Actually, the full attack requires some preparatory steps: entering the building during office hours, surreptitiously sticking the message on the door, and re-entering the office space after everyone has left. But all these actions seem less critical than entering a locked door without holding the key. This

Email addresses: gabriele.lenzini@uni.lu (Gabriele Lenzini), sjouke.mauw@uni.lu (Sjouke Mauw), samir.ouchani@uni.lu (Samir Ouchani)

attack is an example of an intrusion that exploits vulnerabilities in a physical space. It was also possible because it was tolerated that secret reports are left accessible inside the room, optimistically assuming that a locked door was a sufficient protection. This is clearly a deficiency at the level of security policies.

Organizations should be concerned about checking the security of their policies that protect their assets, and the mechanisms guarding the access to the offices should back-up the policies. But checking the effective security of a place is far from being simple. The elements of the game are people, objects which can be moved or stolen and locations that can be locked/unlocked under specific conditions. We call a system with such elements a *Socio-Technical Physical System* (STPS). This term reminds Whitworth and Ahmad's Socio-Technical System [3], which they introduced to describe models about societal and psychological aspects in human computer interactions and problems like mistrust, unfairness and injustice. We consider out of scope any social-science perspective; still we are interesting in human interactions, and by adding "Physical" we intend to stress that the kind of interactions we look at are those with the physical environments.

We sustain that studying the security underneath STPSs, requires methodologies of analysis as reliable as those developed for the analysis of security protocols (*e.g.*, see [4], [5] and [6]). Apparent diversities between the two domains, the technical *vs.* the socio-technical, suggest that the socio-technical security analysis is trickier. Socio-technical systems are made not only of software processes, digital messages and communication channels, but also of people, of physical objects and localities, and of communications that may happen via non-conventional media, such as hand-to-hand, visual, and auditory channels. Despite the increasing interest in the subject, to understand whether formal techniques can help automate the security analysis of STPSs is still on-going research. There is scarcity of tools to figure out whether and how, at what cost and with what probability, an intruder can gain unauthorized access to resources.

Contributions. In this paper, we initiate to develop such an analysis tool. We propose a formalism to model basic elements of STPSs, namely locations, people, and objects. Our formalism contains a rich set of user actions, such as changing locations, manipulating and exchanging objects, locking and unlocking doors and containers. We let actions have a cost and agents act either non-deterministically or probabilistically; actions can be guarded by contextual conditions. We also propose a model of the intruder, as a particular agent able to act maliciously, but according to realistic abilities which it exerts constrainedly depending on physical spaces and costs.

For security analysis, we propose model checking [7, 8] to ensure security in STPSs. We define a map-

ping from STPS models, expressed in the proposed formalism, to the input language of the probabilistic symbolic model checker PRISM [9]. We express security properties in the extended probabilistic computational tree logic (PCTL) [10] and, to overcome the downside of the user-unfriendliness of the logic in expressing natural language security statements, we propose templates for relevant security properties.

Outline. In Section 2 we review the related work. Section 3 describes the formalism that models STPSs and Section 4 details our approach towards security analysis. Experimental results are showed in Section 5. Finally, Section 6 concludes the paper and sketches the future directions.

2. Related Work

In this section we survey the most recent initiatives that deal with modeling, formalizing, and analyzing security in STPSs.

Sommestad *et al.* [11] propose a tool for the analysis of vulnerabilities in computer systems of large organizations and enterprises. The tool, called Cyber-Security Modeling Language (CySeMoL), uses a probabilistic relational model and estimates the probability of success of known attacks which, by experience, are expected to happen. The tool inputs a meta-model of the system, with potential attacks and countermeasures, and estimated attack probabilities of single components. The tool is meant for risk analysis of cyber-security threats, but it is not meant to discover new attacks.

Gunnarsson [12], and Probst and Hansen [13, 14] analyze the robustness of control policies ruling access to an organization's building and its IT network. The analysis highlights what credentials an actor needs in order to reach specific locations and to access specific data. It also shows what he could reach if he had specific credentials. Actors can move within the infrastructure and access the resources therein. The system is modeled in "acKlaim", a calculus of the Klaim family [15], and the analysis consists of a reachability check on locations and actions. This approach is further extended by Probst *et al.* [16] with, amongst others, capabilities and restrictions, allowing an analysis of static and dynamic properties of the system and of the behaviour of actors. Dimkov, Pieters and Hartel [17] also present a dialect of acKlaim, called Portunes. They describe attacks across digital, physical and social security alignments. The attacks are generated by searching through actions and preconditions allowed in the organization. Portunes treats people as physical elements that can do physical actions and are assailable to physical attacks.

The models for the infrastructure and for the actors in [12, 13, 14, 16, 17] have been of inspiration for the model we propose in this paper. But differently, we introduce costs and probabilities. We assume that

not all possible attacks are necessarily feasible: they in fact have costs of execution and probability of success, which are qualities that depend on factors such as time to move from one place to another, effort to act one way instead of another. Our analysis is thus quantitative, making our security properties more interesting because they are expressed in relation to costs and probabilities.

Algarni *et al.*[18] propose a social engineering schema to describe threats in social networks. The schema covers the environment, the attacker, the trick, and the victims. The environment includes privacy settings, friendship and connections, and the content of user profiles. The attacker has been characterised by its ability to understand the victim, and to develop and perform an attack plan. A trick has been identified by the quality of the attack plan and its suitability to the targeted. The victim user supports socio-psychological factors, personality types, demographics variables, and motivations and drives.

Doss and Tejay [19] conduct a study by observing a group of security analysts who detect insider attacks. The goal is to determine how security analysts can use current security detection tools such as log analysis tools and intrusion detection systems to detect insider attacks. This study was conducted based upon the situational research approach from Grounded Theory[1]. Following GT, four categories were created: security monitoring, threat assessment, insider evaluation and goal orientation. The differences on how the groups apply focus of their efforts, their techniques of analyzing activity and different backgrounds can be combined to create an analytical model for detecting insider threats.

To anticipate insider attacks, Greitzer and Hohimer [20] provide a predictive framework that models human behaviors. It integrates cyber data sources, and psychological and motivational factors. The framework adopts functional requirements inspired from the neocortex metaphor. The factors are captured in a hierarchical ontology to interpret employee role, psycho-social, policy violation, and web access patterns.

The limitation of the approaches [18, 19, 20] is that they do not provide any formalization or experimental results. They focus on the high-level strategies for modeling and analyzing security in STPSs.

Meadows and Pavlovic [21] propose a logic framework to reason about permitted and required movements in a physical space, which can be used to analyze security procedures that govern the movement of people. Their procedures define humans controlling hierarchical configurations of physical objects, such as those that arise when we travel, packing our luggage, tickets and documents to satisfy complex

[1] Grounded Theory (GT) is a systematic methodology in the social sciences involving the discovery of theory through the analysis of data.

security and safety requirements. They propose a logic to analyze the procedures, which they use to show the failure of the shoe-screening policy at airports in preventing people to bring disallowed objects in the security area. Compared to our work, this framework is limited to a qualitative specification. Further, we use templates of the temporal logic to express security policies.

Paja *et al.* [22] propose a high-level modeling language for STPSs called STS-ml. It is based on UML diagrams that support social, information, and authorization views. Security requirements are generated based on the security needs between sender and requester actors in terms of constraints. The security analysis uses a disjunctive Datalog solver to identify the conflicting authorisations and the violation of security requirements. This work focuses only on four predefined requirements.

Compared to our approach, none of the commented work covers the probability and costs of actions, formalizes STPS, and measure their security level. Moreover, our security analysis is automatic: we use probabilistic model checking and take advantage from the algorithms built within the tool we use.

3. Formalization

We envisage an STPS as a physical space (*e.g.*, a building) with objects and people. The space is structured in locations (*e.g.*, rooms) accessible via lockable doors. People act in such a space. They lock or unlock doors, enter and exit rooms, manipulate and exchange objects. Each action happens non-deterministically, conditionally, or probabilistically. Executing an action has a cost. In the same space, a malicious entity threatens people's normal workflow.

3.1. STPS : Formal Model

A socio-technical physical system $S \in \mathbf{S}$ is a tuple $S = \langle Phy, Obj, Act, Struc \rangle$. *Phy* models the space, *Obj* models objects, *Act* the actors including the intruder. *Struc* is a hierarchical structure that describes the relations that exist between the entities of an STPS. This structure describes locations that belong to the building, the doors that connect two locations, the location of actors and objects, and further, the objects that an object contains or that an actor holds.

Physical space. *Phy* is a building's infrastructure, its locations and the doors connecting these. It relates keys to the doors and tells if a door is locked or not. Formally, *Phy* is a tuple $\langle L, D, key_D, locked_D \rangle$, where:

- L is a finite set of locations (with elements l, l', etc.).

- D is a finite set of doors (with elements d, d', etc.).
- $key_D : D \mapsto O$ is a partial function that returns the object (*i.e.*, the key) that can lock/unlock a door. key_D is defined only on doors that can be locked. (Objects are defined in the next paragraph)
- $locked_D$ is a predicate. It is true when a door is locked and false when a door is unlocked.

Objects. An object can be a container (*e.g.*, a safe) and containers are lockable. An object, container or not, can be movable, destroyable, or both. Formally, *Obj* is a tuple $\langle O, attr, key_O, locked_O \rangle$, where:

- O is a finite set of objects (with elements o, o', etc.).
- $attr : O \to 2^{\{c,m,d\}}$ returns the set of attributes of an object, which is a subset of $\{c,m,d\}$. Here c stands for be a container, m for being movable, and d for being destroyable.
- $key_O : O \mapsto O$ is a partial function that returns the object (*i.e.*, the key) that can lock/unlock an object. key_O is defined only on objects that are containers which can be locked.
- $locked_O$ is a predicate saying whether an object is locked or not.

Actors. Actors have a behaviour which is constructed from basic actions, composed sequentially or by non-deterministic, conditional, or probabilistic choice. We identify one special actor I as the *intruder*. The main difference between a regular actor and the intruder is that the intruder's behaviour is not restricted by a behavioural specification. Actors' locations, their possessions, the cost of executing actions and the intruder's capabilities are defined later. Formally, *Act* is a tuple $\langle A, I, \Sigma, bv \rangle$ where:

- A is a finite set of actors (with elements a, a', etc.).
- $I \notin A$ is the intruder. We use A_I as a shorthand for the set $A \cup \{I\}$.
- The finite set of basic actions Σ is defined as follows:

  $\Sigma = \{MoveTo(d,l,l'), Lock(d,o), UnLock(d,o), Lock(o,o'), UnLock(o,o'), Put(o,l), Get(o,l),$

  $Put(o,o'), Get(o,o'), Destroy(o), Give(o,a), Rec(o,a) : l,l' \in L$ and $d \in D$ and $o,o' \in O$ and $a \in A\}$

  Informally, "*MoveTo*" means moving through a door from one location to another. "*Lock*" (resp., "*UnLock*") means locking (resp., unlocking) a door or a container with a key. "*Get*" (resp., "*Put*") expresses picking an object up from (resp., putting it down in) a container or a location. "*Destroy*"

stands for destroying an object, and "*Give*" and "*Rec*" for giving and receiving objects between agents.

- $bv_A : A \to \mathscr{L}$ returns the expression that describes the behaviour of an actor.

$\mathscr{L}$, the language of an actor's behavioral expressions, is generated by the following Backus-Naur rule: $B ::= Stop \mid \alpha.B \mid B + B \mid B +_g B \mid B +_p B$, where α is an atomic action in Σ.

Informally, an actor's behavior is built up starting from the basic actions and from a special action *Stop*, expressing inaction. Composite actions are built by sequential composition ".", non-deterministic choice "+", probabilistic choice "$+_p$" (here, p is a probability value), and guarded choice "$+_g$". To simplify the semantical descriptions below, we will assume an equivalence relation on behavioral expressions, satisfying at least the following equalities:

$B_1 + B_2 = B_2 + B_1$, $B_1 +_g B_2 = B_2 +_{\neg g} B_1$, and $B_1 +_p B_2 = B_2 +_{1-p} B_1$.

The guard g is a contextual condition, questioning, for instance, whether an agent possesses the right key, whether a door is unlocked, or whether an object is movable. Formally, a guard is an expression e in the propositional logic language $\mathbb{E}$ defined as follows:

$$\begin{aligned} e &::= \top \mid prop \mid \neg e \mid e \wedge e \\ prop &::= d \in conn(l,l') \mid o \in key_D(d) \mid locked_D(d) \mid loc_O(o) = l \mid c \in attr(o) \mid m \in attr(o) \mid \\ &\quad d \in attr(o) \mid o \in cont_O(o') \mid o \in key_O(o') \mid locked_O(o) \mid loc_A(a) = l \mid o \in cont_A(a) \end{aligned}$$

Here, "$\top$" is the boolean value 'true', '$\wedge$' is conjunction, and '$\neg$' is negation. The proposition $d \in conn(l,l')$ means a door connects l and l', $loc_O(o) = l$ and $loc_A(a) = l$ consider l as a location for o and a, respectively. $o \in cont_O(o')$ and $o \in cont_A(a)$ show if the object o is in a container o' or possessed by a.

Structure. The structure describes the hierarchical relations between locations, objects, actors, and the intruder. It links locations by doors, actors to locations, and objects to locations, to an object (the container), an actor or the intruder. The structure takes the form of a tree extended with special edges defining the connections between locations by doors. Formally, *Struc* is a graph $\langle V, E, C \rangle$, where:

- $V = \{b\} \cup L \cup A_I \cup O$ is the set of vertices, where b is the root that denotes the name of the building;
- $\langle V, E \rangle$ is a directed tree with a root node b, satisfying: $E \subseteq (\{b\} \times L) \cup (L \times (A_I \cup O)) \cup (O \times O)$;

- $C \subseteq L \times D \times L$ is a set of undirected edges between locations, labeled by doors in D.

We formalize the satisfiability of the guard g for $S \in \mathbf{S}$ by the function $[\![\mathbb{E}]\!]_S \to \mathbb{B}$, which evaluates an expression $e \in \mathbb{E}$ in S. Table 1 describes the semantics of the atomic propositions in g. We use S as superscript to describe any function, predicate, or element from S, and we denote by E^+ the transitive closure of the set of edges E. E^+ is the smallest set containing E, satisfying: if $(u,v) \in E^+$ and $(v,w) \in E^+$, then $(u,w) \in E^+$.

$[\![\top]\!]_S \equiv \mathit{true}$	$[\![\mathit{locked}_O(o)]\!]_S \equiv \mathit{locked}_O^S(o)$
$[\![\neg e]\!]_S \equiv \neg[\![e]\!]_S$	$[\![x \in \mathit{attr}(o)]\!]_S \equiv x \in \mathit{attr}^S(o)$ for $x \in \{c,m,d\}$
$[\![e \wedge e']\!]_S \equiv [\![e]\!]_S \wedge [\![e']\!]_S$	$[\![o' \in \mathit{cont}_O(o)]\!]_S \equiv (o,o') \in (E^S)^+$
$[\![d \in \mathit{conn}(l,l')]\!]_S \equiv (l,d,l') \in C^S$	$[\![o \in \mathit{cont}_A(a)]\!]_S \equiv (a,o) \in (E^S)^+$
$[\![o \in \mathit{key}_D(d)]\!]_S \equiv o = \mathit{key}_D^S(d)$	$[\![\mathit{loc}_O(o) = l]\!]_S \equiv (l,o) \in (E^S)^+$
$[\![o \in \mathit{key}_O(o')]\!]_S \equiv o = \mathit{key}_O^S(o')$	$[\![\mathit{loc}_A(a) = l]\!]_S \equiv (l,a) \in E^S$
$[\![\mathit{locked}_D(d)]\!]_S \equiv \mathit{locked}_D^S(d)$	

Table 1: Semantics of guard expressions.

3.2. Labelled Transition System

We model the evolution of an STPS by means of a labelled state transition system $\langle \mathbf{S}, S_0, \Rightarrow \rangle$. Here, $\mathbf{S}$ is the set of all STPS states, $S_0 \in \mathbf{S}$ is a given state that we call the initial state, and for a set of labels Γ, $\Rightarrow \ \subseteq (\mathbf{S} \times \Gamma \times \mathbf{S})$ is the transition relation.

Labels. Labels are tuples of the form $\Gamma \subseteq \Sigma_\ell \times \mathbb{R}^+ \times [0,1]$, where Σ_ℓ are transition names, $\mathbb{R}^+$ are costs, and $[0,1]$ are probability values. The names in Σ_ℓ carry the action being executed, the actor (possibly the intruder) as well as the objects and subjects of the action.

$$\begin{aligned}\Sigma_\ell = \{&\mathit{moving}_x(d,l,l'), \mathit{locking}_x(d,o), \mathit{unlocking}_x(d,o), \mathit{locking}_x(o,o'), \mathit{unlocking}_x(o,o'), \mathit{dropping}_x(o,l),\\ &\mathit{grasping}_x(o,l), \mathit{placing}_x(o,o'), \mathit{withdrawing}_x(o,o'), \mathit{destroying}_x(o), \mathit{giving}_x(a,o), \mathit{taking}_x(a,o),\\ &\mathit{exchanging}_x(a,a',o), \mathit{stealing}_x(a,o), \mathit{slipping}_x(a,o), \mathit{lockpicking}_x(l), \mathit{lockpicking}_x(o) \mid\\ &l,l' \in L \text{ and } d \in D \text{ and } o,o' \in O \text{ and } a \in A \text{ and } x \in A_I\}\end{aligned}$$

The real values in $\mathbb{R}^+$ denote the cost of the transition. We assume that costs are established by a function $\mathit{Cost} : \Sigma_\ell \to \mathbb{R}^+$. Costs depend on all the elements of the transition: the action, the action's subject and its object. So, for instance, the cost of an actor moving from one location to another depends (or may depend) on the actor and on the two locations. In particular, the cost of the intruder opening a

door with a key may be different from the cost of opening the door by picking the lock. The probabilistic values denote the probability of the transition.

Transition relation. The transition relation $\Rightarrow$ defines how STPS states change in consequence of what the actors and the intruder do. It is the smallest relation that satisfies the transition rules given in the reminder of this paragraph. We write each transition in its infix form $S \xRightarrow{(\ell,c,p)} S'$, where the states are represented by the relevant part of their graphs. The edge labels and the nodes in the graphs express elements of the state that relate to them. We display only those elements that express a condition for the occurrence of the transition or that change due to the transition. Besides, we often simplify the transition label. We omit the cost c, as it is $Cost(\ell)$ and can be easily retrieved when it evaluates to true, and we omit the probability value when this is 1. This simplification regards all rules, except those expressing guarded (GC) and probabilistic (PC) choices. We present the transition rules of actors in Tables 2, 3, and 4, and we describe the rules of the intruder in Table 5.

Actors. Table 2 shows the transition rules that describe the actions of an actor. Rule (M) describes an actor a moving from location l to a connected location l' via an unlocked door d. Rule (LD) concerns locking and (UD) concerns unlocking a door d with the key k. Rule (LO) describes the locking and (UO) unlocking of a container.

Table 3 shows how an STPS changes because of an actor dropping an object o in the location where he stands (PL), or putting it inside another container which is within the actor's grasp (PO). Symmetrically, rules GO and GL describe the action of picking an object up from an unlocked container or a location, respectively. Rule D describes how the STPS changes after an actor destroys an object. The exchanging rule (EX) expresses synchronization of giving and receiving an object o. As a result, the object becomes possession of the receiving party. In specifications, we will often write $Rec(x,a)$ for $Rec(o_1,a) + \cdots + Rec(o_m,a)$ to express that the recipient is willing to receive any object. The cost of an exchange is defined as the sum of the costs of the two synchronized actions, where c denotes the cost of giving and c' denotes the cost of receiving the object.

Finally, Table 4 shows the rules for the choice operators, respectively non-deterministic choice (NC), guarded choice (GC) and probabilistic choice (PC). B_1, B_2, B'_1, and B'_2 are assumed to be behavioural expressions of the same actor. We write $a[B]$ to express a behavior of an actor a in a given state S.

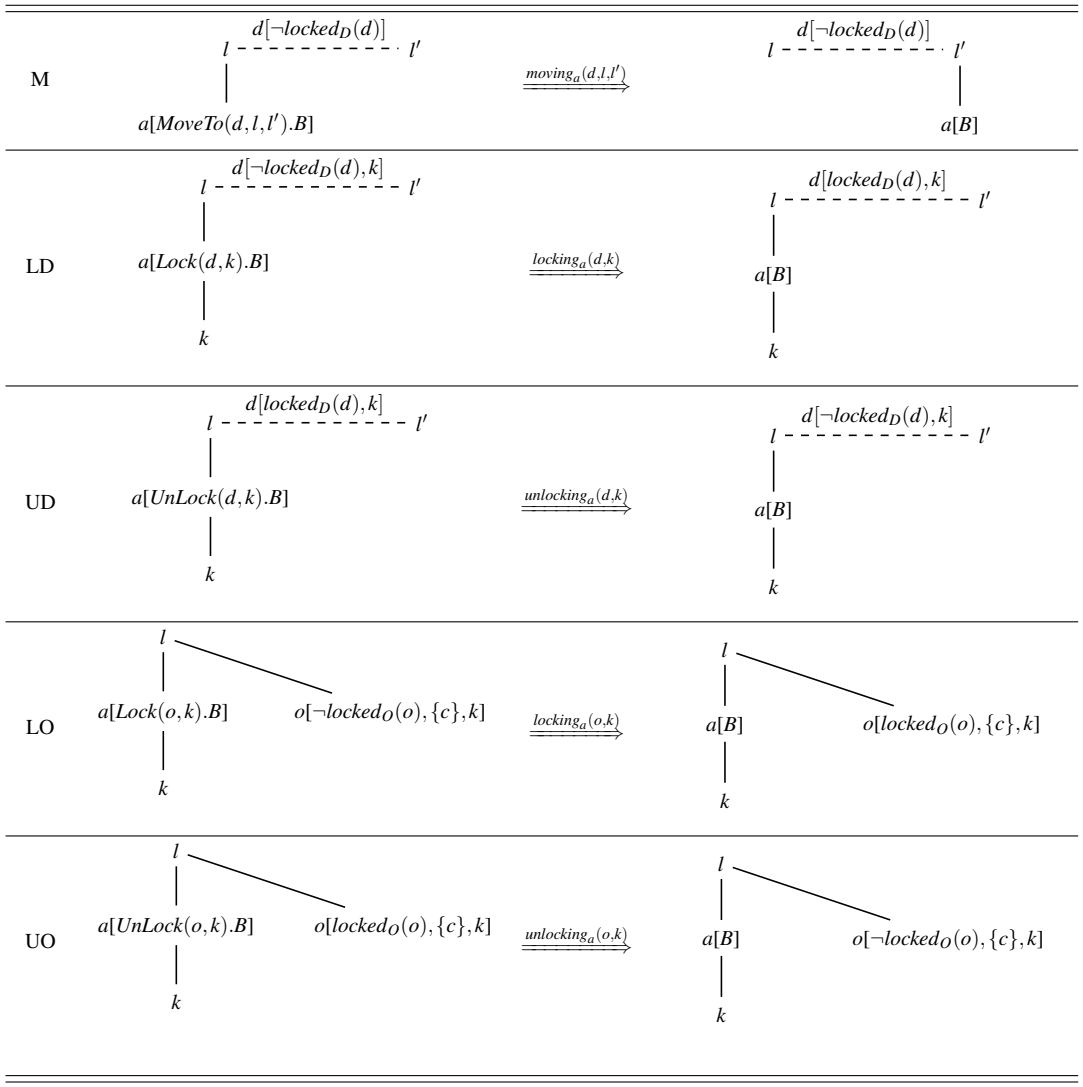


Table 2: The STPS semantic rules - Part 1.

Intruder. The intruder is modeled similar to any other actor: he is in a certain room, possibly holding some objects. However, while a regular actor's behavior is described by a behavioral expression, the intruder is not restricted in this way. The capabilities of the intruder are given by means of a set of semantical rules, which describe, for instance, that the intruder can open locks without possession of the key. An example of a set of such semantical intruder rules is given in Table 5. We consider this particular set of rules as a parameter of our modeling language, which can be replaced by any stronger or weaker intruder model. The rules in Table 5, describe a more or less realistic, relatively weak, intruder.

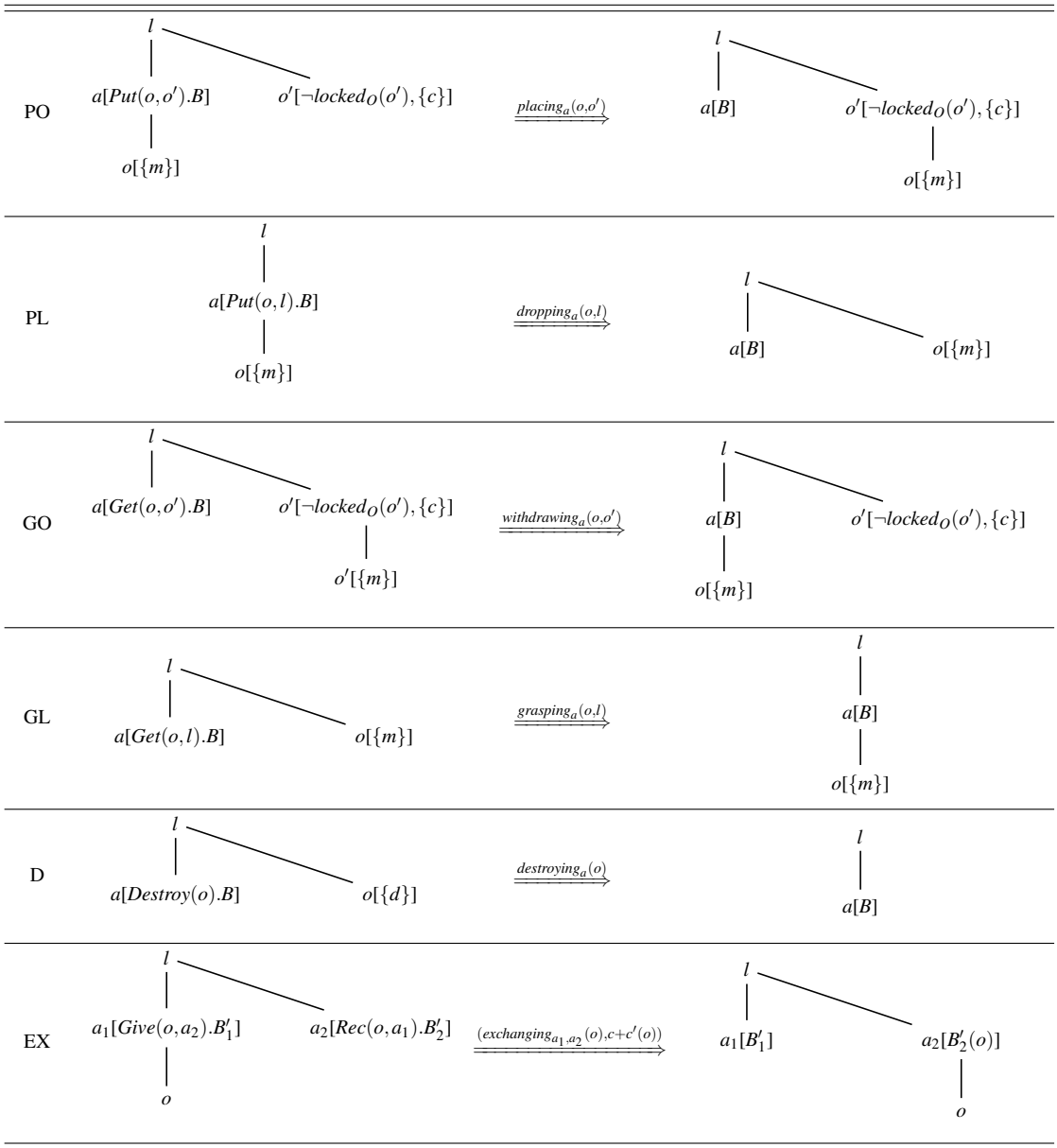


Table 3: The STPS semantic rules - Part 2.

Rules LD_I, UD_I, LO_I, UO_I express that the intruder can lock and unlock doors and containers without possessing the key. Rules S_I and T_I state that the intruder can steal objects from an agent or slip objects into an agent's pockets. Rules G_I and R_I describe the situation where the intruder impersonates a regular agent in the exchange of objects. In addition to these intruder-specific activities, the intruder has the same capabilities as those of regular agents (Tables 2 and 3), such as moving (rule M) and object manipulation

NC	$\dfrac{S[B_1] \xrightarrow{(\ell,c,p)} S'[B_1{}']}{S[B_1 + B_2] \xrightarrow{(\ell,c,p)} S'[B_1{}']}$	GC	$\dfrac{S[B_1] \xrightarrow{(\ell,c,p)} S'[B_1'] \quad [\![g]\!]_S}{S[B_1 +_g B_2] \xrightarrow{(\ell,c,p)} S'[B_1{}']}$	PC	$\dfrac{S[B_1] \xrightarrow{(\ell,c,q)} S'[B_1]}{S[B_1 +_p B_2] \xrightarrow{(\ell,c,p\times q)} S'[B_1{}']}$

Table 4: The STPS semantic rules - Part 3.

(rules PO, PL, GO, GL, D). The intruder variants of these rules are straightforward and will not be displayed.

Clearly, the chosen set of intruder capabilities still has some strong limitations. First of all, the intruder's actions are restricted to the physical infrastructure provided, which means, e.g., that he will have to use doors to move from one location to another, that he needs to be in a room to pick up something from that room and that he cannot move an unmovable object, as he cannot destroy an indestructible object. Second, the intruder is not able to change the physical infrastructure, so he is not able to create new doors or locations. We repeat that these limitations are not based on the assumption that our model should only consider relatively weak adversaries and that the semantics rules can be easily extended.

4. Security Analysis

This section details our approach to analyze STPS security. An overview of the involved steps is depicted in Figure 1. An STPS model is mapped to a PRISM program. PRISM programs are recalled in Section 4.1, and the algorithm defining the stepwise construction of a PRISM program from an STPS model is listed in Section 4.2. The diagram also shows the use of security template expressions which define relevant security properties and are instantiated as extended PCTL formulas. The instantiation is described in Section 4.3. Finally, PRISM checks the satisfiability of security properties in the considered model, and produces the verification result in terms of probability and cost.

4.1. PRISM Program

A PRISM program is a set of *modules*, each having a countable set of boolean or integer, local, variables. A module's local state is fully defined by the evaluation of its local variables, while the program's global state is defined by the evaluation of all variables.

The behavior of a module is defined by a set of *probabilistic commands*, of *Dirac commands*, or both. Probabilistic commands have the form $[a]\ g \rightarrow p_1 : u_1 + ... + p_m : u_m$, where p_i are probabilities ($p_i \in]0,1[$

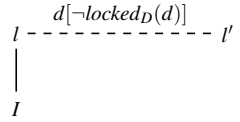
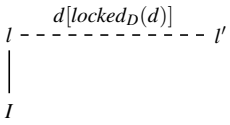
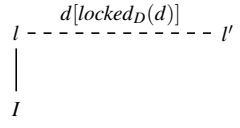
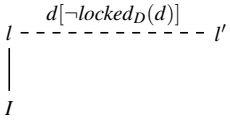
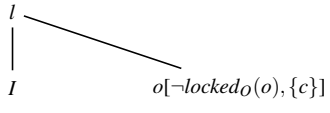
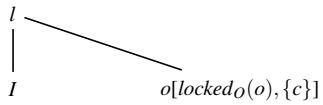
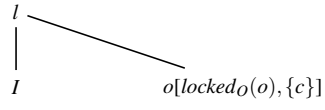
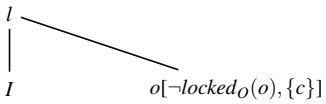
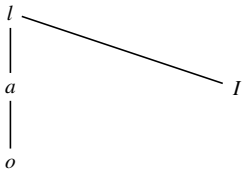
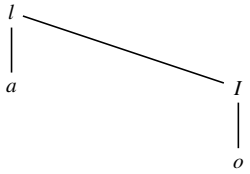
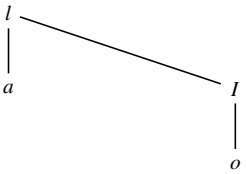
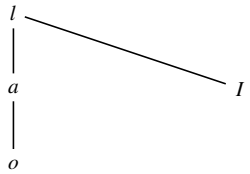
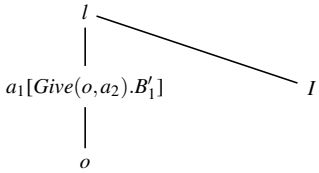
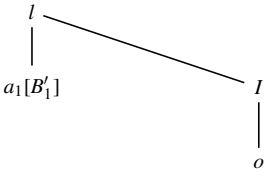
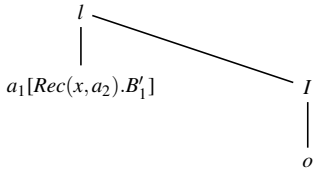
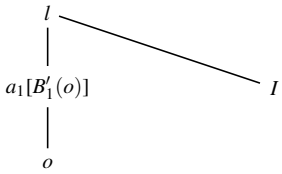
LD_I		$\xRightarrow{locking_I(d)}$	
UD_I		$\xRightarrow{unlocking_I(d)}$	
LO_I		$\xRightarrow{locking_I(o)}$	
UO_I		$\xRightarrow{unlocking_I(o,k)}$	
S_I		$\xRightarrow{stealing_I(o,a)}$	
T_I		$\xRightarrow{slipping_I(o,a)}$	
G_I		$\xRightarrow{(exchanging_{a_1,I}(o),c+c')}$	
R_I		$\xRightarrow{(exchanging_{a_1,I}(o),c+c'(o))}$	

Table 5: The STPS semantic rules - Part 4.

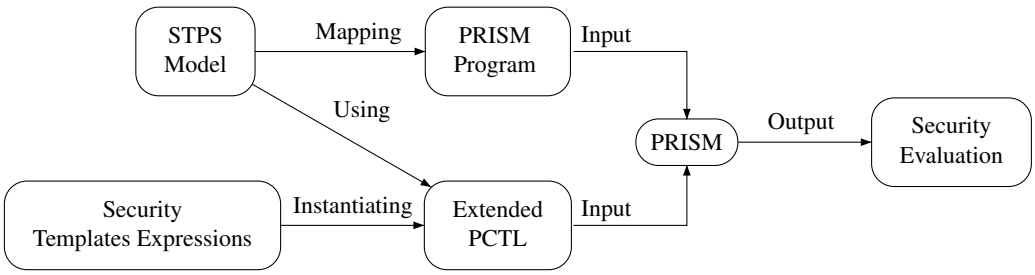


Figure 1: Our methodology to evaluate the security of Socio-Technical Physical Systems.

and $\sum_i p_i = 1$), a is a label, g is a boolean predicate on local and global variables (*i.e.*, a *guard*), and u_i are *updates* for variables. An update, written as $(v'_j = val_j)\&\cdots\&(v'_k = val_k)$, reassigns variables v_i with values val_i. The meaning of a probabilistic command is the following: for action a, if the guard g is satisfied, an update u_i is enabled with a probability p_i. Dirac commands have the form $[a]\ g \rightarrow u$, which means that for a, if the guard g is true, the update u is enabled with probability 1. The action a is a label. The guard is an expression consisting of both local and global variables, and the propositional logic operators. Syntactically, a module M is delimited by two keywords: the module head "`module` M", and the module termination "`endmodule`".

We can model costs in PRISM. In a PRISM program, say P, costs (called *rewards* in PRISM's jargon) are specified in a separate module. Syntactically, a reward module R is delimited by keywords "`rewards` R" and "`endrewards`". A reward contains one or more *reward items*, called *state reward* or *transition reward*. A state reward has form $g : r$, where g is guard, and r is an real-valued algebraic expression on variables and constant. It means that the reward r is associated to any state satisfying g. A transition reward has the form $[a]\ g : r$. It expresses that the transitions labelled a, from states satisfying g, are acquiring the reward r.

Modules can be composed and they can communicate, for which PRISM uses the composition operators from the process algebra CSP (*e.g.*, see [23]). Full synchronization of modules M_1, and M_2 (on all shared action) is written as $M_1||M_2$, their parallel interface synchronization, which is limited to the set of shared actions $\{a,b,\cdots\}$, is written as $M_1|[a,b,\cdots]|M_2$. Modules M_1 and M_2's interleaving, that is their asynchronous parallel composition, is written as $M_1|||M_2$.

4.2. From STPS Models to PRISM Programs

We encode an STPS model into an equivalent PRISM program. We chose Markov Decision Processes as the PRISM formalism because they conform better to the STPS semantics given by the transition rules defined in Section 3.

The encoding is specified by five functions, namely Ψ_A, Ψ_O , Ψ_L, Ψ_I and Ψ_C. Given an STPS model $S \in \mathbf{S}$ ($\mathbf{S}$ is the set of STPS models) they return a PRISM program $\mathrm{P} \in \mathbb{P}$ ($\mathbb{P}$ is the set of PRISM programs). Specifically, each function maps a fragment of the STPS model into a PRISM module. Function Ψ_A encodes the actors and the transition rules that have an effect on them. Ψ_O encodes the objects and the transition rules that have an effect on objects. Ψ_L encodes the physical space, locations and doors, and the transition rules affecting them. These three functions together encode the structure which links actors, objects, locations and doors and the changes that operate on them by the STPS transition rules. Besides, Ψ_I encodes the intruder and the transition rules of the intruder's actions, and Ψ_C encodes the transition costs. The final PRISM program is the composition, by synchronization, of the five modules. Our mapping models locations $l \in L$ as integers and assumes the set of PRISM variables given in Table 6.

variable	meaning
$\mathtt{x}_o$	$o \in O$
$\mathtt{c}_o$	$attr(o) = c$
$\mathtt{m}_o$	$attr(o) = m$
$\mathtt{d}_o$	$attr(o) = d$
$\mathtt{l}_a$ (type integer)	$(l,a) \in E$
$\mathtt{l}_o$ (type integer)	$(l,o) \in E$
$\mathtt{l}_\mathtt{I}$ (type integer)	$(l,I) \in E$

variable	meaning
d_{ij}	$(l_i,d,l_j) \in C$
a_o	$(a,o) \in E$
$o_{o'}$	$(o,o') \in E$
$\mathtt{I}_{o'}$	$(I,o') \in E$
$\mathtt{C}_d$	$locked_D(d)$
$\mathtt{C}_o$	$locked_O(o)$
$\mathtt{K}_{do}$	$key_D(d) = o$
$\mathtt{K}_{oo'}$	$key_O(o) = o'$

Table 6: PRISM variables and their meaning. Unless stated differently, all variables are boolean. Metavariable a ranges over A, o over O, and d over D.

The function Ψ_A, which produces the suitable PRISM commands related to the actors, is presented in Listing 1. There is a command for each transition rule whose conclusion refers to actors. The action labels of each command correspond to labels of STPS transitions. For instance, the PRISM command's label $M_{ij}^{a,d}$ corresponds to the STPS model's transition label $moving_a(d,l_i,l_j)$ (line 3). The guard of the command depends on the premise of the transition rule that corresponds to the command. The updates express the conclusions of the transition rule.

So, for instance, transition rule M in Table 2, whose premises are (a,l_i) and (l_i,d,l_j) (implicitly from

the edges) and $d[\neg locked_D(d)]$ (explicitly stated), maps to guard $(\mathtt{l}_a = i) \wedge d_{ij} \wedge \neg \mathtt{C}_d$. Its conclusion, (a, l_j), maps to update $\mathtt{l}'_a = j$. To take another example, transition rule LD in Table 2, whose premises are (a, l_i), (a,k) and (l_i, d, l_j) (implicitly from the edges) and $d[\neg locked_D(d)]$ and $key_D(d) = k$ (explicitly stated), maps to guard $(\mathtt{l}_a = i) \wedge a_k \wedge d_{ij} \wedge \neg \mathtt{C}_d \wedge \mathtt{K}_{dk}$. The rule's conclusion, its part concerning the actor, is empty so it maps to the neutral update $\mathtt{l}'_a = i$, which changes nothing. The mapping concerning the part of the rule's conclusion that relates to the change of status of the door, from unlocked to locked, is specified in Ψ_L.

In Listing 1 we use the function $\diamondsuit$ that adds a boolean to the guard of a command, for example: $g' \diamondsuit [a]g \to u \equiv [a]g \wedge g' \to u$. This is used for the commands related to the guarded and probabilistic choices (line 16 and line 17, respectively).

1 $\Psi_A : \mathbf{S} \to \mathbb{P}$

2 $\Psi_A(S) = \forall b \in B_a$, Case (b) **of**

3 $MoveTo(d,l_i,l_j).B \Rightarrow$ **in** $\{[M^{a,d}_{ij}]((\mathtt{l}_a = i) \wedge (d_{ij}) \wedge \neg \mathtt{C}_d) \to (\mathtt{l}'_a = j);\} \cup \Psi_A(\mathbf{S})[B_a/B] \cup \Psi_O(\mathbf{S}) \cup \Psi_C(\mathbf{S})$ **end**

4 $Lock(d,k).B \Rightarrow$ **in** $\{[L^a_{d,k}]((a_k) \wedge (\mathtt{l}_a = i) \wedge (\mathtt{K}_{dk}) \wedge (d_{ij}) \wedge \neg \mathtt{C}_d) \to (\mathtt{l}'_a = i);\} \cup \Psi_A(\mathbf{S})[B_a/B] \cup \Psi_L(\mathbf{S}) \cup \Psi_C(\mathbf{S})$ **end**

5 $UnLock(d,k).B \Rightarrow$ **in** $\{[U^a_{d,k}]((a_k) \wedge (\mathtt{l}_a = i) \wedge (\mathtt{K}_{dk}) \wedge (d_{ij}) \wedge \mathtt{C}_d) \to (\mathtt{l}'_a = i);\} \cup \Psi_A(\mathbf{S})[B_a/B] \cup \Psi_L(\mathbf{S}) \cup \Psi_C(\mathbf{S})$ **end**

6 $Lock_a(o,k).B \Rightarrow$ **in** $\{[L^a_{o,k}]((a_k) \wedge (\mathtt{l}_a = \mathtt{l}_o) \wedge (\mathtt{K}_{ok}) \bigwedge_{a' \neq a} \neg(a'_o) \wedge \neg \mathtt{C}_o) \to (\mathtt{l}'_a = \mathtt{l}_o);\} \cup \Psi_A(\mathbf{S})[B_a/B] \cup \Psi_O(\mathbf{S}) \cup \Psi_C(\mathbf{S})$ **end**

7 $UnLock(o,k).B \Rightarrow$ **in** $\{[U^a_{o,k}]((a_k) \wedge (\mathtt{l}_a = \mathtt{l}_o) \wedge (\mathtt{K}_{ok}) \wedge \mathtt{C}_o \bigwedge_{a' \neq a} \neg(a'_o)) \to (\mathtt{l}'_a = \mathtt{l}_o);\} \cup \Psi_A(\mathbf{S})[B_a/B] \cup \Psi_O(\mathbf{S}) \cup \Psi_C(\mathbf{S})$ **end**

8 $Put_a(o,o').B \Rightarrow$ **in** $\{[P^a_{o,o'}]((\mathtt{c}_{o'}) \wedge (\mathtt{m}_o) \wedge (a_o) \wedge (\mathtt{l}_a = \mathtt{l}_{o'}) \wedge \neg \mathtt{C}_{o'} \bigwedge_{a' \neq a} \neg(a'_{o'})) \to ((a_o)' = \bot);\} \cup \Psi_A(\mathbf{S})[B_a/B] \cup \Psi_O(\mathbf{S}) \cup \Psi_C(\mathbf{S})$ **end**

9 $Put_a(o,l_i).B \Rightarrow$ **in** $\{[P^a_{o,l}]((\mathtt{m}_o) \wedge (a_o) \wedge (\mathtt{l}_a = i)) \to ((a_o)' = \bot);\} \cup \Psi_A(\mathbf{S})[B_a/B] \cup \Psi_O(\mathbf{S}) \cup \Psi_C(\mathbf{S})$ **end**

10 $Get(o,o').B \Rightarrow$ **in** $\{[G^a_{o,o'}]((\mathtt{c}_{o'}) \wedge (\mathtt{m}_o) \wedge (o'_o) \wedge (\mathtt{l}_a = \mathtt{l}_{o'}) \wedge \neg \mathtt{C}_{o'} \bigwedge_{a' \neq a} \neg(a'_{o'})) \to ((a_o)' = \top);\} \cup \Psi_A(\mathbf{S})[B_a/B] \cup \Psi_O(\mathbf{S}) \cup \Psi_C(\mathbf{S})$ **end**

11 $Get(o,l_i).B \Rightarrow$ **in** $\{[G^a_{o,l}]((\mathtt{m}_o) \wedge (\mathtt{l}_o = i) \wedge (\mathtt{l}_a = i)) \to ((a_o)' = \top);\} \cup \Psi_A(\mathbf{S})[B_a/B'] \cup \Psi_O(\mathbf{S}) \cup \Psi_C(\mathbf{S})$ **end**

12 $Destroy(o).B \Rightarrow$ **in** $\{[D^a_o]((\mathtt{d}_o) \wedge (a_o) \wedge (\mathtt{l}_a = \mathtt{l}_o) \bigwedge_{a' \neq a} \neg(a'_o)) \to ((a_o)' = \bot);\} \cup \Psi_A(\mathbf{S})[B_a/B] \cup \Psi_O(\mathbf{S}) \cup \Psi_C(\mathbf{S})$ **end**

13 $Give(o,a').B \Rightarrow$ **in** $\{[EX^a_{o,a'}]((a_o) \wedge (\mathtt{l}_a = \mathtt{l}_{a'})) \to ((a_o)' = \bot);\} \cup \Psi_A(\mathbf{S})[B_a/B] \cup \Psi_O(\mathbf{S}) \cup \Psi_C(\mathbf{S})$ **end**

14 $Rec(o,a').B \Rightarrow$ **in** $\{[EX^a_{o,a'}]((a'_o) \wedge (\mathtt{l}_a = \mathtt{l}_{a'})) \to ((a_o)' = \top);\} \cup \Psi_A(\mathbf{S})[B_a/B'] \cup \Psi_O(\mathbf{S}) \cup \Psi_C(\mathbf{S})$ **end**

15 $B_1 + B_2 \Rightarrow$ **in** $\Psi_A(\mathbf{S})[B_1 + B_2/B_1]) \cup \Psi_A(\mathbf{S})[B_1 + B_2/B_2])$ **end**

16 $B_1 +_g B_2 \Rightarrow$ **in** $g \diamondsuit \Psi_A(\mathbf{S})[B_1 +_g B_2/B_1]) \cup \neg g \diamondsuit \Psi_A(\mathbf{S})[B_1 +_g B_2/B_2])$ **end**

17 $B_1 +_p B_2 \Rightarrow$ **in** $\{[+_p](i_1 = \bot) \wedge (i_2 = \bot) \Rightarrow p : (i'_1 = \top) + 1 - p : (i'_2 = \top);\} \cup i_1 \diamondsuit \Psi_A(\mathbf{S})[B_1 +_p B_2/B_1]) \cup \neg i_2 \diamondsuit \Psi_A(\mathbf{S})[B_1 +_p B_2/B_2])$ **end**

18 Otherwise **in end**

Listing 1: Generating the Actors Modules.

The function Ψ_O shown in Listing 2 generates the commands of the objects modules. These commands express the objects' behaviours described in the conclusions of STA rules. The variable $\mathtt{l}_o$ is of type integer describing the location of the object o that is initialised by its first location and $\mathtt{x}_o$ is a boolean expressing the existence of the object o. Also, the boolean $o_{o'}$ expresses the ownership of the object o' by the container o. All commands generated by Ψ_O synchronize with the similar ones obtained by Ψ_A.

1 $\Psi_O\colon\ \mathbf{S} \to \mathbb{P}$

2 $\Psi_O(S)\ =\ \forall b \in B_a,\ \text{Case}\ (b)\ \mathbf{of}$

3 $MoveTo(d,l_i,l_j).B \Rightarrow\ \mathbf{in}\ \{[M_{ij}^{a,d}]((\mathtt{l}_a = i) \wedge (d_{ij}) \wedge \neg\mathtt{C}_d \wedge a_o) \to ((\mathtt{l}_o)' = j);\}\ \mathbf{end}$

4 $Lock_a(o,k).B \Rightarrow\ \mathbf{in}\ \{[L_{o,k}^a]((a_k) \wedge (\mathtt{l}_a = \mathtt{l}_o) \wedge (\mathtt{K}_{ok}) \bigwedge_{a' \neq a} \neg(a'_o) \wedge \neg\mathtt{C}_o) \to ((\mathtt{C}_o)' = \top);\}\ \mathbf{end}$

5 $UnLock(o,k).B \Rightarrow\ \mathbf{in}\ \{[U_{o,k}^a]((a_k) \wedge (\mathtt{l}_a = \mathtt{l}_o) \wedge (\mathtt{K}_{ok}) \wedge \mathtt{C}_o \bigwedge_{a' \neq a} \neg(a'_o)) \to ((\mathtt{C}_o)' = \bot);\}\ \mathbf{end}$

6 $Put_a(o,o').B \Rightarrow\ \mathbf{in}\ \{[P_{o,o'}^a]((\mathtt{c}_{o'}) \wedge (\mathtt{m}_o) \wedge (a_o) \wedge (\mathtt{l}_a = \mathtt{l}_{o'}) \wedge \neg\mathtt{C}_{o'} \bigwedge_{a' \neq a} \neg(a'_{o'})) \to ((o'_o)' = \top);\}\ \mathbf{end}$

7 $Put_a(o,l_i).B \Rightarrow\ \mathbf{in}\ \{[P_{o,l}^a]((\mathtt{m}_o) \wedge (a_o) \wedge (\mathtt{l}_a = i)) \to ((l_o)' = i);\}\ \mathbf{end}$

8 $Get(o,o').B \Rightarrow\ \mathbf{in}\ \{[G_{o,o'}^a]((\mathtt{c}_{o'}) \wedge (\mathtt{m}_o) \wedge (o'_o) \wedge (\mathtt{l}_a = \mathtt{l}_{o'}) \wedge \neg\mathtt{C}_{o'} \bigwedge_{a' \neq a} \neg(a'_{o'})) \to ((o'_o)' = \bot);\}\ \mathbf{end}$

9 $Get(o,l_i).B \Rightarrow\ \mathbf{in}\ \{[G_{o,l}^a]((\mathtt{m}_o) \wedge (\mathtt{l}_o = i) \wedge (\mathtt{l}_a = i)) \to ((\mathtt{l}_o)' = \mathtt{l}_a);\}\ \mathbf{end}$

10 $Destroy(o).B \Rightarrow\ \mathbf{in}\ \{[D_o^a]((\mathtt{d}_o) \wedge (\mathtt{l}_a = \mathtt{l}_o) \bigwedge_{a' \neq a} \neg(a'_o)) \to ((\mathtt{x}_o)' = \bot);\}\ \mathbf{end}$

11 $Give(o,a').B \Rightarrow\ \mathbf{in}\ \{[EX_{o,a'}^a]((a_o) \wedge (\mathtt{l}_a = \mathtt{l}_{a'})) \to ((\mathtt{l}_o)' = \mathtt{l}_{a'});\}\ \mathbf{end}$

12 $Rec(o,a').B \Rightarrow\ \mathbf{in}\ \{[EX_{o,a'}^a]((a'_o) \wedge (\mathtt{l}_a = \mathtt{l}_{a'})) \to ((\mathtt{l}_o)' = \mathtt{l}_{a'});\}\ \mathbf{end}$

Listing 2: Generating the Objects Modules.

The function Ψ_L presented in Listing 3 produces the PRISM commands related to the status of doors. The boolean proposition $\mathtt{C}_{d_{ij}}$ shows if the door d_{ij} between locations i and j is locked or not.

1 $\Psi_L\colon\ \mathbf{S} \to \mathbb{P}$

2 $\Psi_L(S)\ =\ \forall b \in B_a,\ \text{Case}\ (b)\ \mathbf{of}$

3 $Lock(d,k).B \Rightarrow\ \mathbf{in}\ \{[L_{d,k}^a]((a_k) \wedge (\mathtt{l}_a = i) \wedge (\mathtt{K}_{dk}) \wedge (d_{ij}) \wedge \neg\mathtt{C}_d) \to ((\mathtt{C}_d)' = \top);\}\mathbf{end}$

4 $UnLock(d,k).B \Rightarrow\ \mathbf{in}\ \{[U_{d,k}^a]((a_k) \wedge (\mathtt{l}a = i) \wedge (\mathtt{K}_{dk}) \wedge (d_{ij}) \wedge \mathtt{C}_d) \to ((\mathtt{C}_d)' = \bot);\}\mathbf{end}$

Listing 3: Generating the Objects Modules.

The function Ψ_C defined in Listing 4 produces the cost commands by calling the function *Cost*. The command in line 3 is a generic command that returns the cost related to the action α where g_α is the guard to execute α.

1 $\Psi_C\colon\ \mathbf{S} \to \mathbb{P}$

2 $\Psi_C(S)\ =\ \forall b \in B_a,\ \text{Case}\ (b)\ \mathbf{of}$

3 $\alpha.B_a \Rightarrow\ \mathbf{in}\ \{[\alpha](g_\alpha) : Cost(\alpha);\}\ \mathbf{end}$

Listing 4: Generating the Costs Modules.

4.3. Security Properties

Since the scope of our formalization is to verify socio-technical security properties on STPS models, we comment in this section what properties can be of relevance and how to express them in such a way that they can be checked by running PRISM. A formalism that is able to express all the factors

that our STPS models describe, that is paths along locations, paths of actions, propositions on state variables, probabilities of occurrence of events and of sequences of events, and their costs is the *extended* PCTL [10]. Formulas ϕ in such a logic are generated by the following BNF grammar:

$$\begin{aligned} \phi &::= \top \mid ap \mid \phi \wedge \phi \mid \neg\phi \mid \mathrm{P}_{\bowtie p}[\psi] \mid \mathrm{R}_{\bowtie r}[F\phi] \\ \psi &::= \mathrm{X}\phi \mid \phi \mathrm{U} \phi \mid \phi \mathrm{U}^{\leq k} \phi \end{aligned}$$

Here, $k \in \mathbb{N}$, $r \in \mathbb{R}^+$, $p \in [0,1]$, and $\bowtie \in \{<, \leq, >, \geq\}$. A state formula can be "ap", an atomic proposition, or any propositional expression built from "ap". $\mathrm{P}_{\bowtie p}[\psi]$, called *probabilistic path predicate*, returns true if the probability to satisfy the *path formula* ψ is $\bowtie p$. The *cost predicate* $\mathrm{R}_{\bowtie r}[\phi]$ returns true if the cost to satisfy ϕ is $\bowtie r$. Here, F is the temporal logic operator *eventually*. A path formula is built from the typical temporal operators *next* (X), *until* (U), and *bounded until* ($\mathrm{U}^{\leq k}$). As usual, other logic operators can be derived from the basic operators. Namely:

- $\bot \equiv \neg\top$, $\phi \vee \phi' \equiv \neg(\neg\phi \wedge \neg\phi')$, $\phi \rightarrow \phi' \equiv \neg\phi \vee \phi'$, and $\phi \leftrightarrow \phi' \equiv \phi \rightarrow \phi' \ \wedge \ \phi' \rightarrow \phi$.
- $F\phi \equiv \ \top \ U \ \phi$, $F^{\leq k}\phi \equiv \ \top \ U^{\leq k} \ \phi$, $G\phi \equiv \ \neg(F\neg\phi)$, and $G^{\leq k}\phi \equiv \ \neg(F^{\leq k}\neg\phi)$ where $k \in \mathbb{N}$.
- $\mathrm{P}_{\geq p}[G\phi] \equiv \mathrm{P}_{\leq 1-p}[F\neg\phi]$.

Besides, the *extended* PCTL has two more probabilistic operators, Pmin and Pmax, and two more cost operators, Rmin and Rmax. These operators can be used within a path or a state formulas to express the minimum (resp. maximum) probability or cost.

In order to simplify the specification of STPS requirements, we define three PCTL expression templates. Assuming that σ is a predicate logic expression built from three atomic propositions, namely: ($\mathtt{a}_o$) (*i.e.*, agents a holds object o), ($\mathtt{l}_a = i$) (*i.e.*, agent a is in location i), and ($\mathtt{l}_o = i$) (*i.e.*, object o is in location i), we define:

1. $\phi := F\sigma$
2. $\phi' := \mathrm{Rmin}[\phi]$
3. $\phi'' := \mathrm{Pmax}[\phi]$

Using the templates we can express reachability properties in a socio-technical sense, such as the property that eventually an object is found in a location, that an actor, or the intruder, reaches a room, or

that he will get in possession of an object. They can be used to check for possibility of intrusions or of thefts. The quantitative expressions ϕ' and ϕ'' are used to quantify, in term of minimal cost and maximum probability, a security property. We show how to use such properties in the next section where we develop a proof-of-concept use case.

5. Case Study

Let 'E' be the name of a building in a small institution. At lunch time, the 'E' building's workers leave their office doors open or closed, but without locking them. There are no policies or security primitives that enforce them to lock doors and to prevent security incidents from insiders. Further, the 'E' building has two gates to enter or exit, which are always open from 6 am to 6 pm. During this time, no access card/key is needed to enter the building. Under these assumptions, we use our framework to measure the probability and the cost to get access to an unauthorized location and to steal objects by an intruder. The building's infrastructure is depicted in Figure 2. It has nine locations, labeled $l_0, \cdots, l_8$, where l_8 denotes outside the building. Doors d_{08} and d_{18} are the entry/exit points to/from the building that connect l_0 with l_8 and l_1 with l_8, respectively. We consider at least two mobile objects o_1 and o_2 located in l_2 and l_6, respectively.

Formal Model

First, we develop the infrastructure tree, and we describe the behavior of an actor a. Then, we show the generated PRISM code for the actor, the intruder, and the objects. Figure 3 captures part of the graph model of the infrastructure shown in Figure 2. It shows the initial values of the actor a, the intruder I, and the objects o_1 and o_2. The behavior of a is expressed as follows:
$MoveTo(d_{0\,8}, l_8, l_0) \cdot MoveTo(d_{0\,2}, l_0, l_2) \cdot Get(o_1, l_2) \cdot MoveTo(d_{2\,5}, l_2, l_5) \cdot MoveTo(d_{5\,6}, l_5, l_6) \cdot Put(o_1, l_6) \cdot$
$Get(o_2, l_6) \cdot MoveTo(d_{6\,5}, l_6, l_5) \cdot MoveTo(d_{5\,2}, l_5, l_2) \cdot MoveTo(d_{2\,0}, l_2, l_0) \cdot MoveTo(d_{0\,8}, l_0, l_8)$.

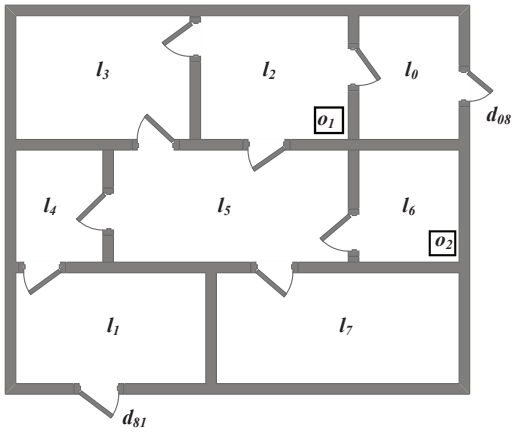


Figure 2: Floor plan of the Infrastructure.

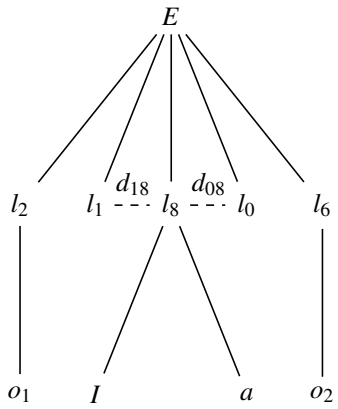


Figure 3: The Infrastructure Model.

PRISM Model

For the performance assessment of the system, the whole system is encoded into PRISM; and Listing 5 shows the code fragment for the actor a, the intruder I, both objects o_1 and o_2, and the cost module. In Listing 5, the module "Actor$_a$" (lines 25-40) describes the behavior of the actor a and the module "Intruder" (lines 3-23) describes the behavior of the intruder. To add more actors, a user instantiates the module "Actor$_a$" by renaming only the actor's local variables. Also, the modules "Object$_{o_1}$" (lines 41-47) and "Object$_{o_2}$" (lines 48-54) describe the behaviours of objects "o_1" and "o_2", respectively. The module "Cost" (lines 55-60) assigns the cost to the actions. In the module "Intruder", the probabilistic command in line 7 defines the behavior of the intruder when he is outside the building and the set of commands in lines 9-14 describe the possible moving actions when he is in the location l_4. In addition, lines 16-22 define two actions, take and put an object, respectively. For the objects $o1$ and $o2$, the commands in lines 43, 44, 50, and 51 describe the moving of objects, otherwise they stay at the same locations (lines 45, 46, 52, and 53). Finally, the module "cost" defines the cost to access to location l_2 (line 57), to possess an object (lines 58 and 59) by a possessor, and the cost to possess an object by the intruder. Otherwise, the cost in line 56 attributes the value 1 to any other action.

```
mdp
const int step;
module Intruder
  l_I: [0..8] init 8; // Locations
  I_o1: bool init ⊥; I_o2: bool init ⊥;
  // Outside the building
  [M_8] (l_I=8)->0.2:(l_I'=8)+0.4:(l_I'=0)+0.4:(l_I'=1);
  //The intruder's behavior at l_4
  [M_42] (l_I=4) ⇒ (l_I'=2);
  [M_43] (l_I=4) ⇒ (l_I'=3);
  [M_44] (l_I=4) ⇒ (l_I'=4);
  [M_45] (l_I=4) ⇒ (l_I'=5);
  [M_46] (l_I=4) ⇒ (l_I'=6);
  [M_47] (l_I=4) ⇒ (l_I'=7);
  ....// Taking or stealing an object
  [T_o1] (l_I=l_o1) ⇒ (I'_o1=⊤);
  [T_o2] (l_I=l_o2) ⇒ (I'_o2=⊤);
  ....// Put/slip an object
  [P_o1,l_i] (l_I=l_i) & (I_o1) ⇒ (I'_o1=⊥);
  [S_o1,a] (l_I=l_i) & (I_o1) ⇒ (I'_o1=⊤);
  [P_o2,l_i] (l_I=l_i) & (I_o2) ⇒ (I'_o2=⊥);
  [S_o2,a] (l_I=l_i) & (I_o2) ⇒ (I'_o2=⊤);
endmodule

module Actor_a
  l_a: [0..8] init 8; // Locations
  a_o1: bool init ⊥; a_o2: bool init ⊥;
  [M_80] (l_a=8) ⇒ (l'_a=0);
  [M_02] (l_a=0) ⇒ (l'_a=2);
  [G_a1] (l_a=2) ⇒ (a'_o1=⊤);
```

Listing 5: The PRISM Code Fragment of the System.

```
  [M_25] (l_a=2)&(a_o1) ⇒ (l'_a=5);
  [M_56] (l_a=5) ⇒ (l_a'=6);
  [P_o1] (l_a=6) &(a_o1)⇒ (a'_o1=⊥);
  [T_o2] (l_a=6) &(l_o2) &(¬I_o2)⇒(a'_o2=⊤);
  [M_65] (l_a=6) &(a_o2)⇒ (l'_a=5);
  [M_52] (l_a=5) &(a_o2)⇒ (l'_a=2);
  [M_20] (l_a=2) &(a_o2)⇒ (l'_a=0);
  [M_08] (l_a=0) &(a_o2)⇒ (l'_a=8);
  [M_88] (l_a=8) &(a_o2)⇒ (l'_a=8);
endmodule
module object_o1
  l_o1 : [0..8] init 2; // Locations
  [M_a1] (a_o1)⇒ (l'_o1=l_a);
  [M_I] (I_o1)⇒ (l'_o1=l_I);
  [M_a1] (¬a_o1)⇒ (l'_o1=l_o1);
  [M_I] (¬I_o1)⇒ (l'_o1=l_o1);
endmodule
module object_o2
  l_o2 : [0..8] init 6;
  [M_a1] (a_o2)⇒ (l'_o2=l_a);
  [M_I] (I_o2)⇒ (l'_o2=l_I);
  [M_a1] (¬a_o2)⇒ (l'_o2=l_o2);
  [M_I] (¬I_o2)⇒ (l'_o2=l_o2);
endmodule
rewards cost
true:1;
[M_02] (l_a=2) : 2;
[] (a_o1) | (a_o2): 3;
[] (I_o1) | (I_o2): 3;
endrewards
```

Listing 6: PRISM Code - Continuation.

Specification and Verification of Properties

In order to validate the functionality of the specified system, we verify four security properties, which are instantiations of the templates developed in Section 4.3.

Property 1. "What is the maximum probability for the intruder I to possess both objects o_1 and o_2?" This property is instantiated from ϕ'' and expressed as follows: $Pmax=?[\top\ U \leq step\ (l_{o_2} = l_8)\&(l_{o_1} = l_8)\&(I_{o_1})\&(I_{o_2})]$. The variable $step$ is the number of steps to reach the state that satisfies: $(l_{o_2} = l_8)\&(l_{o_1} = l_8)\&(I_{o_1})\&(I_{o_2})$ where the propositions $l_{o_1} = l_8$ and $l_{o_2} = l_8$ mean that both objects are

outside the building. The propositions I_{o_1} and I_{o_2} show that the intruder possesses both objects. The verification result of this property is presented by Figure 4. It shows the convergence of the probability evaluation of *Property 1* from 0 at step 11 to 0.026 after 28 steps, then it increases up to 1 after 50 steps. This result shows that the opportunity to steal an object is increasing while the intruder is in the building.

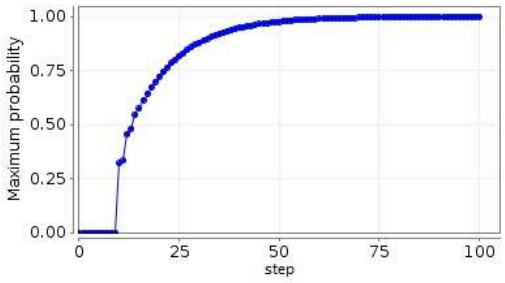


Figure 4: The verification result of *Property 1*.

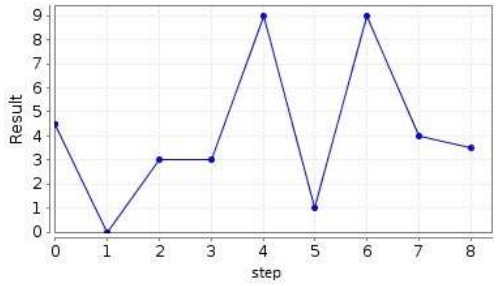


Figure 5: The verification result of *Property 2*.

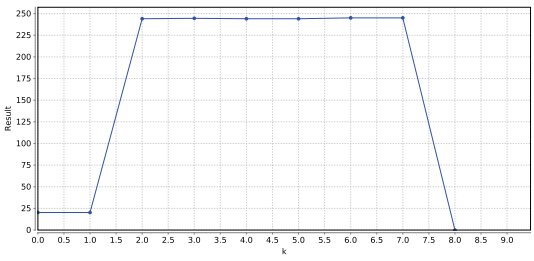


Figure 6: The verification result of *Property 3*.

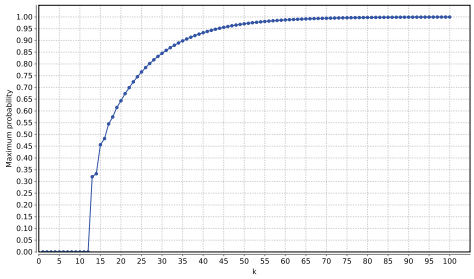


Figure 7: The results of *Property 4*.

Property 2. This property measures the minimum cost to get the object o_1 from any location in the building and it is expressed from the formula ϕ' by: $Rmin = ?[FI_{o_1}\{l_I = i\}]$ where $l_I = i$ means the initial location of the intruder. After verification, Figure 5 shows that the cost obtained from the satisfiability of *Property 2* is 9 from locations l_4 and l_6, and 3 from locations l_2 and l_3, and 4 from l_7 and l_8. From this result, we observe that the cost increases more when the intruder is far from the object location.

Property 3. This property evaluates the minimum cost to get the object o_1 and go outside the building. Similarly to *Property 2*, it is instantiated from ϕ' and it is expressed by: $Rmin = ?[F(I_{o_1} \& l_I = 8)\{l_I = i\}]$. The verification results depicted in Figure 6 show that the reward obtained from the satisfiability of *Property 3* is 240 for locations $l_2, \cdots, l_7$, 24 for l_1, and 0 for l_0 and l_8. The obtained results show the highest cost needed to possess an object and going outside the building from a given location.

Property 4. "What is the maximum probability to possess an object by either an actor or the intruder at any location?" This property is instantiated from ϕ'' and expressed by: $Pmax = ?[\top U \leq step((I_{o_2} \& a_{o_1}) \vee$

$(I_{o_1} \& a_{o_2}))]$. Figure 7 shows that the probability results of this property converges to 1 after 55 steps. We observe that the probability to possess an object is close to one for both the actor and intruder, since they stay at the building.

6. Conclusion

We have defined a framework for automatic security analysis of socio-technical physical systems. Our formalism models aspects of a socio-technical physical space such as its spatial infrastructure, its objects –among which doors, locking keys, containers and assets–, and agents that move across the infrastructure and manipulate objects. Actions have a cost and decisions are guided by probabilities or by contextual conditions. The semantics is rich and allows to express security properties about whether attacks exist with maximal likelihood and minimal cost. These security properties can be verified automatically. We map our model to the specification language of the probabilistic model checker PRISM, and express our security properties in the Probabilistic Computational Tree Logic, which can be processed by PRISM. The effectiveness and efficiency of our approach is illustrated with a simple case study.

The work done has been conceived to allow several further developments. We plan to extend our model with *information*, allowing for the modeling of *digital objects* and *knowledge*. This would make our model richer, able to express socio-technical physical and information systems. Our notion of intruder will be also extended with capabilities mirroring actions documented in social-engineering and incidents reports. Further, we plan to develop a security policy language, in order to enrich our framework with security policies. From a more theoretical point of view, we plan to develop correctness and soundness proofs of our mapping functions. Finally, we intend to implement the framework as a software tool that fully supports all steps in the design and analysis of socio-technical physical systems.

7. Acknowledgments

The research leading to the results presented in this work received funding from the Fonds National de la Recherche Luxembourg, project ”Socio-Technical Analysis of Security and Trust”, C11/IS/1183245, STAST, and the European Commissions Seventh Framework Programme (FP7/2007-2013) under grant agreement number 318003 (TREsPASS).

References

[1] B. Medsger, The Burglary: The Discovery of J. Edgar Hoover's Secret FBI, Knopf Doubleday Publishing Group, 2014.

[2] B. Schneier, 1971 Social Engineering Attack, https://www.schneier.com/blog/archives/2014/02/1971_social_eng.html.

[3] B. Whitworth, A. Ahmad, The Social Design of Technical Systems: Building Technologies for Communities, The Interaction Design Foundation, 2013, https://www.interaction-design.org/books/the_social_design_of_technical_systems.html.

[4] A. Bauer, J. Juerjens, Security Protocols, Properties, and Their Monitoring, in: Fourth International Workshop on Software Engineering for Secure Systems (SESS'08), ACM, New York, USA, 2008, pp. 33–40.

[5] C. Cremers, S. Mauw, Operational Semantics and Verification of Security Protocols, Information Security and Cryptography, Springer Verlag, 2012.

[6] P. Ryan, S. Schneider, The Modelling and Analysis of Security Protocols: The CSP Approach, Addison-Wesley, 2000.

[7] E. M. Clarke, O. Grumberg, D. A. Peled, Model Checking, The MIT Press, 1999.

[8] C. Baier, J. P. Katoen, Principles of Model Checking, The MIT Press, 2008.

[9] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: Verification of Probabilistic Real-time Systems, in: 23rd International Conference on Computer Aided Verification (CAV'11), Vol. 6806 of LNCS, Springer Verlag, 2011, pp. 585–591.

[10] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, Automated Verification Techniques for Probabilistic Systems, in: Formal Methods for Eternal Networked Software Systems, Vol. 6659 of LNCS, Springer Verlag, 2011, pp. 53–113.

[11] T. Sommestad, M. Ekstedt, H. Holm, The Cyber Security Modeling Language: A Tool for Assessing the Vulnerability of Enterprise System Architectures, IEEE Systems Journal 7 (3) (2013) 363–373.

[12] D. Gunnarsson, Static Analysis of the Insider Problem, Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Lyngby, Denmark (2007).

[13] C. W. Probst, R. R. Hansen, An Extensible Analysable System Model, Information Security Technical Report 13 (4) (2008) 235–246.

[14] C. Probst, R. Hansen, Analysing access control specifications, in: Fourth Int. IEEE Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'09), IEEE, 2009, pp. 22–33.

[15] R. De Nicola, G. Ferrari, R. Pugliese, KLAIM: a Kernel Language for Agents Interaction and Mobility, IEEE Transactions on Software Engineering 24 (5) (1998) 315–330.

[16] C. Probst, R. Hansen, F. Nielson, Where Can an Insider Attack?, in: Formal Aspects in Security and Trust (FAST'06), Vol. 4691 of LNCS, Springer Verlag, 2007, pp. 127–142.

[17] T. Dimkov, W. Pieters, P. Hartel, Portunes: Representing Attack Scenarios Spanning through the Physical, Digital and Social Domain, in: Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (ARSPA-WITS'10), Vol. 6186 of LNCS, Springer Verlag, 2010, pp. 112–129.

[18] A. Algarni, Y. Xu, T. Chan, Y.-C. Tian, Social Engineering in Social Networking Sites: Affect-based Model, in: 8th International Conference for Internet Technology and Secured Transactions (ICITST'13), IEEE, 2013, pp. 508–515.

[19] G. Doss, G. Tejay, Developing Insider Attack Detection Model: A Grounded Approach, in: IEEE International Conference on Intelligence and Security Informatics (ISI'09), 2009, pp. 107–112.

[20] F. L. Greitzer, R. E. Hohimer, Modeling Human Behavior to Anticipate Insider Attacks, Journal of Strategic Security 4 (2) (2011) 25–48.

[21] C. Meadows, D. Pavlovic, Formalizing Physical Security Procedures, in: Security and Trust Management (STM'12), Vol. 7783 of LNCS, Springer Verlag, 2013, pp. 193–208.

[22] D. Fabiano, G. Paolo, M. John, Adaptive Socio-Technical Systems: a Requirements-based Approach, Requirement Engineering 18 (1) (2013) 1–24.

[23] C. Hoare, Communicating Sequential Processes, Prentice Hall International, Incorporated, 1985.