Portunes: analyzing multi-domain insider threats

Trajce Dimkov, Wolter Pieters, Pieter Hartel

Distributed and Embedded Security Group
University of Twente, The Netherlands
{trajce.dimkov, wolter.pieters, pieter.hartel}@utwente.nl

Abstract. The insider threat is an important problem in securing information systems. Skilful insiders use attack vectors that yield the greatest chance of success, and thus do not limit themselves to a restricted set of attacks. They may use access rights to the facility where the system of interest resides, as well as existing relationships with employees. To secure a system, security professionals should therefore consider attacks that include non-digital aspects such as key sharing or exploiting trust relationships among employees. In this paper, we present Portunes, a framework for security design and audit, which incorporates three security domains: (1) the security of the computer system itself (the digital domain), (2) the security of the location where the system is deployed (the physical domain) and (3) the security awareness of the employees that use the system (the social domain). The framework consists of a model, a formal language and a logic. It allows security professionals to formally model elements from the three domains in a single framework, and to analyze possible attack scenarios. The logic enables formal specification of the attack scenarios in terms of state and transition properties.

Keywords: insider threat, physical security, security awareness, security model.

1 Introduction

Malicious insiders are a serious threat to organizations. Motivated by greed or malice, insiders can disrupt services, modify or steal data, or cause physical damage to the organization. Protecting assets from an insider is challenging [1] since insiders have knowledge of the security policies in place, have certain privileges on the systems and are trusted by colleagues. An insider may use the knowledge of the security policies to avoid detection and use personal credentials or social engineer colleagues to carry out an attack. An example of such a scenario is the road apple attack [2], where an insider tricks an employee into plugging a malicious dongle into a server located in a physically restricted area. Thus, the environment in the organization where the insider operates spans all three security domains, physical security, digital security and security awareness of the employees.

This research is supported by the Sentinels program of the Technology Foundation STW, applied science division of NWO and the technology programme of the Ministry of Economic Affairs under project number TIT.7628.

Current formal models for modeling insider threats [3,4,5] assume the insider uses only digital means to achieve an attack. Therefore, these models do not look into mobility of people and devices nor social interactions between people, and focus only on modeling the security of a network or a host. For example, there is a lot of research that focuses on modeling and analyzing network and host configurations to generate, analyze and rank attack scenarios using attack graphs [6].

Assuming that the insider uses only digital means to achieve an attack leaves an essential part of the environment of interest not captured in the security models. Indeed, a study performed by the National Threat Assessment Center in the US (NTAC) [7] shows that 87% of the attacks performed by insiders require no technical knowledge and 26% use physical means or the account of another employee as part of the attack. Thus, a whole family of attacks, digitally-enabled physical attacks and physically-enabled digital attacks [8], in which the insider uses physical, digital and social means to compromise the asset cannot be presented nor analyzed. Representing all three security domains in a single formalism is challenging because the domains have different properties which makes them hard to integrate.

The contribution of this paper is Portunes[1], a framework which integrates all three security domains in a single environment, hereby enabling analysis of multi-domain insider threats. Portunes consists of a model, a language and a logic. The model is a high-level abstraction of the environment focusing on the relations between the three security domains. It provides a conceptual overview of the environment that is easy to understand by the user. The language is at a relatively low level of abstraction, close to the enforcement mechanisms. The language is able to describe security policies and mechanism which span the three security domains. We use the language to automatically generate attack scenarios across these domains. The Portunes logic is able to formally express properties of models and model evolutions, which are used to specify undesired goals and to select subset of attacks. The Portunes framework is implemented using the Groove model checker, which allows graphical representation of a Portunes model, easy implementation of the semantics of the Portunes language and checking properties using the Portunes logic.

The rest of the paper is structured as follows. Section 2 formalizes the Portunes model and Section 3 formalizes the Portunes language. Section 4 gives a logic for presenting properties of a Portunes model. We use the road apple attack as a running example of the scenarios Portunes is designed to represent. Section 5 discusses implementation issues and section 6 gives an overview of related work which contributes to the design of Portunes. The final section concludes the paper and identifies future work.

[1] After Portunes, the Roman god of keys

2 Portunes

This section presents the Portunes framework. We first present the requirements which Portunes needs to satisfy and the motivation behind some of the design decisions. Based on the requirements, we formally define the Portunes model and the Portunes language. To show the expressiveness of the framework, we use an instance of the road apple attack as an example.

2.1 Requirements and motivation

The three security domains focus on different aspects of security. Physical security restricts access to buildings, rooms and objects. Digital security is concerned with access control on information systems. Finally, security awareness of employees focuses on resistance to social engineering, and is achieved through education of the employees.

Representing all three security domains in a single formalism is challenging. Firstly, the appropriate abstraction level needs to be found. A too low level of abstraction for each domain (down to the individual atoms, bits or conversation dynamics) makes the representation complicated and unusable. However, abstracting away from physical spaces, data and relations between people might omit details that contribute to an attack. Thus, a model integrating multiple security domains needs to be expressive enough to present the relevant details of an attack in each security domain. In previous work [9], we provided the basic requirements for an integrated security model to be expressive enough to present detailed attacks. Briefly, an integrated security model should be able to present the data of interest, the physical objects in which the data resides, the people that manipulate the objects and the interaction between data, physical objects and people.

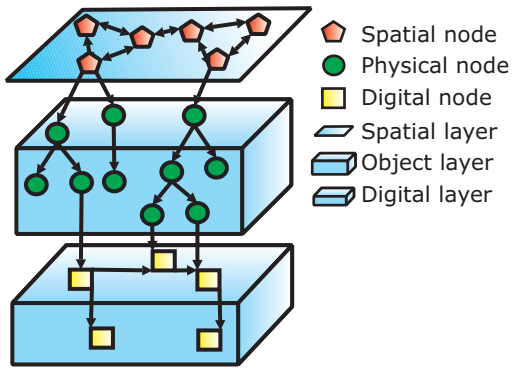

Fig. 1. Graphic presentation of Portunes

Secondly, the domains have different properties making them hard to integrate. For example, mobility of digital data is not restricted by its locality as is

the case with objects in the physical domain. Likewise, physical objects cannot be reproduced as easily as digital data. An additional requirement for Portunes is to restrict interactions and states which are not possible in reality. For example, it is possible to put a laptop in a room, however, putting a room in a laptop is impossible; a person can move only to a neighboring location, while data can move to any location; data can be easily copied, while the reproduction of a computer requires assembling of other objects or materials.

2.2 The Portunes model

To present the different properties and behavior of elements from physical and digital security, the Portunes model stratifies the environment of interest in three layers: spatial, object and digital. The spatial layer presents the facility of the organization, including rooms, halls and elevators. The object layer consists of objects located in the facility of the organization, such as people, computers and keys. The digital layer presents the data of interest. Stratification of the environment in three distinct layers allows specification of actions that are possible only in a single layer (copying can only happen for digital entities) or between specific layers (a person can move data, but data cannot move a person).

The Portunes model abstracts the environment of an organization in a stratified graph and restricts the edges between layers to reflect reality. A node abstracting a location, such as an elevator or a room, belongs to the spatial layer L and it is termed a spatial node. A node abstracting a physical object, such as a laptop or a person, belongs to the object layer O and it is termed an object node. A node abstracting data, such as an operating system or a file, belongs to the digital layer D and it is termed a digital node. The edges between spatial nodes denote a neighbor relation and all other edges in the model denote a containment relation. The ontology used in Portunes is given in Figure 2. An edge (n, m) between two spatial nodes means n *is a neighbor of* m. This is a symmetric relation where the direction of the edge is not important. For all other nodes, an edge (n, m) means that node n *contains* node m; this is an asymmetric relation.

layer	node	edge
spatial	location	neighbors
		contains
object	physical object	contains
		contains
digital	data	contains

Fig. 2. The ontology of Portunes

The above statements are illustrated in Figure 1 and formalized in the following definition.

Definition 1. *Let $G = (Node, Edge)$ be a directed graph and $\mathcal{D} : Node \rightarrow Layer$ a function mapping a node to Layer $= \{L, O, D\}$. A tuple $(G, \mathcal{D})$ is a Portunes model if it satisfies the following invariants $C(G, \mathcal{D})$:*

1. *Every object node can have only one parent.*
   $\forall n \in Node : \mathcal{D}(n) = O \rightarrow indegree(n) = 1$
2. *One of the predecessors of an object node must be a spatial node.*
   $\forall n \in Node : \mathcal{D}(n) = O \rightarrow \exists m \in Node : \mathcal{D}(m) = L \wedge \exists \langle m, ..., n \rangle$*; where* $\langle m, ..., n \rangle \in Edge^+$ *denotes a finite path from m to n, and $Edge^+$ is a finite set of finite paths.*
3. *There is no edge from an object to a spatial node.*
   $\nexists (n, m) \in Edge : \mathcal{D}(n) = O \wedge \mathcal{D}(m) = L$
4. *There is no edge from a digital to an object node.*
   $\nexists (n, m) \in Edge : \mathcal{D}(n) = D \wedge \mathcal{D}(m) = O$
5. *A spatial and a digital node cannot be connected.*
   $\nexists (n, m) \in Edge : (\mathcal{D}(n) = D \wedge \mathcal{D}(m) = L) \vee (\mathcal{D}(n) = L \wedge \mathcal{D}(m) = D)$
6. *The edges between digital nodes do not generate cycles.*
   $\nexists \langle n, ..., m \rangle \in Edge^+ : \mathcal{D}(n) = ... = \mathcal{D}(m) = D \wedge n = m$

The intuition behind the invariants is as follows. An object node cannot be at more than one place, thus an object node can have only one parent (1). An object node is contained in a known location (2). An object node cannot contain any spatial objects (3) (for example, a laptop cannot contain a room) nor can a digital node contain an object node (4) (for example, a file cannot contain a laptop). A spatial node cannot contain a digital node and vice versa (5), and a digital node cannot contain itself (6).

Theorem 1. *A graph $G = (Node, Edge)$ in a Portunes model $(G, \mathcal{D})$ can have cycles only in the spatial layer:*
$\exists \langle n, ..., m \rangle \in Edge^+ : n = m \rightarrow \mathcal{D}(n) = ... = \mathcal{D}(m) = L$

Proof. The proof is presented in the appendix.

Example 1: Road apple attack To show how Portunes can be used for representing insider threats across domains, we will use the example of the road apple attack [2]. In this attack, an insider installs malicious software on a dongle. Then the insider gives the dongle to an employee. The giving of the dongle can happen directly, where the insider convinces the employee to take the dongle, or the insider can simply spread multiple infected dongles around the vicinity of the employee's working place in hope that the employee will find and take one. After the employee takes the dongle, she plugs it in a computer which is in a secure location but connected to the Internet. Then the malicious software installs itself to the computer and transfers the sensitive data from the employee's computer

to a remote location. In this paper we will formalize the attack in the following steps. First, the insider convinces the employee to take the dongle by abusing her trust (social domain). Then, the employees goes to a server in a restricted area and plugs in the dongle (physical domain). Finally, the malicious software from the dongle transfers the sensitive data to a remote server (digital domain).

To describe the attack, the environment in which the attack takes place needs to include information from all three security domains. Concerning physical security, the organization has a restricted area where a server with sensitive data resides. Additionally there is a public area where employees can socialize. Regarding the digital domain, the sensitive data on the server is isolated from the rest of the network, making the data accessible only locally. The security awareness of the employees is such that they trust each other enough to share office material (for example: CDs and dongles).

$$\mathcal{D}(hall) = \mathcal{D}(secureRoom) = \mathcal{D}(world) = L$$
$$\mathcal{D}(remoteServer) = \mathcal{D}(insider) = \mathcal{D}(employee) = \mathcal{D}(secureServer) = \mathcal{D}(dongle) = O$$
$$\mathcal{D}(serverData) = \mathcal{D}(rootkit) = D$$

Fig. 3. The function $\mathcal{D}$ for the road apple attack environment

The segregation of the nodes among the layers is presented in Figure 3. The nodes *hall*, *secureRoom* and *world* are spatial nodes, *serverData* and *rootkit* are digital nodes. All other nodes are object nodes. An abstraction of the Portunes model is graphically presented in Figure 4. The spatial nodes are presented as red pentagons, the object nodes as green circles and the digital nodes as yellow squares. The edges in the graph present the relationship between the nodes. For example, the hall is neighboring the secure room and the secure room contains a secure server which in turn contains server data.

In Section 3 we define the language that formally specifies the model and in Section 3.2 we will revisit the example and show *how* the road apple attack takes place using the formal specification. In Section 4 we will show how to present properties on the road apple example formally.

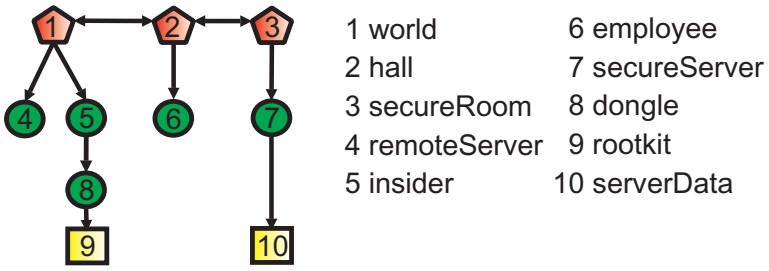

Fig. 4. Graph of the road apple attack environment

3 The Portunes language

In the previous section, we defined a graph-based model to present the facilities of an organization, the objects in a facility and the data of interest. This model is on a conceptual level, and it simplifies the presentation of the environment to the user. In this section we introduce the Portunes language. The language consists of nodes, processes and actions, where a node in the Portunes language represents a node in the Portunes model. The language formally describes the model and makes it more suitable to describe and analyze the enforcement mechanisms as well as to formally specify the interaction between the nodes.

The language captures two interactions, *mobility* and *delegation*. By making all nodes first class citizens, every node can move. For example, a node representing an insider can move through the organization and collect keys, which increase the initial privileges of the insider. Similarly, a spatial node representing an elevator can move between floors in a building. In the Portunes language, a delegator node can delegate a task to a delegatee node. By delegation here we refer to the act of granting the delegatee additional privileges to carry out a task on behalf of the delegator.

The above two interactions, mobility and delegation, are restricted by the invariants from Definition 1 and by the security policies associated with each node. Policies on nodes from the spatial and object layer represent the physical security. These policies restrict the physical access to spatial areas in the facility and the objects inside the spatial areas. Policies on nodes from the digital layer represent the digital security of the organization and focus on access control on the data of interest. In the Portunes language people can interact with other people. Policies on people give the social aspect of the model, or more precisely, they define under which circumstances a person *trusts* another person.

3.1 Overview of Klaim

The Portunes language is inspired by the Klaim family of languages. Klaim (Kernel Language for Agent Interaction and Mobility) is an experimental kernel programming language designed to model and program distributed concurrent applications with code mobility [10]. The syntax of Klaim is presented in Figure 5.

Klaim relies on the concept of distributed tuple space. A tuple space is a multiset of tuples. A tuple t is a container of information which can be either an actual value such as an expression e, process P, or a locality ℓ formal fields such as value variables $!x$, process variables $!X$ and locality variables $!u$. An example of a tuple is: (5, "person", !var), where 5 and "person" are expressions and !var is a value variable. Tuples are anonymous and Klaim uses pattern matching to select tuples from a tuple space.

A node contains one tuple space and processes which can run in parallel. A node can be accessed through its address. There are two kinds of addresses: sites s and localities ℓ. Sites are absolute identifiers through which nodes can be uniquely identified within a net and localities are symbolic names for nodes and

N	::=		Node
	$\mid$	0	Empty net
	$\mid$	$s ::_\rho P$	Single node
	$\mid$	$N_1 \parallel N_2$	Net composition
P	::=		Process
	$\mid$	nil	Null process
	$\mid$	$act.P$	Action prefixing
	$\mid$	$P_1 + P_2$	Choice
	$\mid$	$P_1 \mid P_2$	Parallel composition
	$\mid$	X	Process variable
	$\mid$	$A\langle \tilde{P}, \tilde{\ell}, \tilde{e} \rangle$	Process invocation

act ::= out(t)@ℓ | in(t)@ℓ | read(t)@ℓ | eval(P)@ℓ | newloc(u)

t ::= $e \mid P \mid \ell \mid !x \mid !X \mid !u \mid t_1, t_2$

Fig. 5. Syntax of the Klaim language

have a relative meaning depending on the node where they are interpreted. Localities are associated with sites through allocation environments ρ, represented as partial functions on each node.

Klaim processes may run concurrently and can perform five basic operations over nodes. Three of them, $in(t)@\ell$, $read(t)@\ell$, $out(t)@\ell$ are used to manipulate the tuples, $newloc(u)$ creates a new node and $eval(P)@\ell$ spawns a process P for execution at node ℓ

3.2 Syntax of the Portunes language

As with other members of the Klaim family, the syntax of the Portunes language consists of nodes, processes and actions. The Portunes language lacks the tuple spaces and the actions associated with tuple spaces, which are present in the Klaim family of languages, and focuses on the connections between nodes. This is because connectivity is the main interest from the perspective of security modeling.

The syntax of the Portunes language is shown in Figure 6. A single *node* $l ::_s^\delta P$ consists of a name $l \in \mathcal{L}$, where $\mathcal{L}$ is a universe of node names, a set of node names $s \in 2^{\mathcal{L}}$, representing nodes that are connected to node l , an access control policy δ and a process P. The relation between the graph of the Portunes model and the expressions in the Portunes language is intuitive: a node l in the graph represents a node with name l in the language, an edge (l, l') in the graph connects l to a node name $l' \in s$ of the node $l ::_s^\delta P$. Thus, the node name uniquely identifies the node in the model, while the set s defines which other nodes the node *contains* or *is a neighbor of*. These two relations identify the relative location of each element in the environment. A net is a composition of nodes.

N	$::=$		Node
	$\mid$	$l ::_s^\delta P$	Single node
	$\mid$	$N_1 \parallel N_2$	Net composition
P	$::=$		Process
	$\mid$	nil	Null process
	$\mid$	$P_1 \mid P_2$	Process composition
	$\mid$	$a^l.P_1$	Action prefixing
a	$::=$		Action
	$\mid$	$login(l)$	Login
	$\mid$	$logout(l)$	Logout
	$\mid$	$eval(P)@l$	Spawning

Fig. 6. Syntax of the Portunes language

A *process* P is a composition of actions. Namely, nil stands for a process that cannot execute any action and $a^l.P_1$ for the process that executes action a using privileges from node $l \in \mathcal{L}$ and then behaves as P_1. The label l identifies a node from where the privileges originate, and it is termed the origin node. The structure $P_1|P_2$ is for parallel composition of processes P_1 and P_2. A process P represents a task. A node can perform a task by itself or delegate the task to another node. Recursive and mutually recursive process definitions are not allowed in the Portunes language. Thus, every behavior described using the language has to have an end.

An *action* a is a primitive which manipulates the nodes in the language. There are three primitives, $login(l)$, $logout(l)$ and $eval(P)@l$. The actions $login(l)$ and $logout(l)$ provide the mobility of a node, by manipulating the set s. The action $eval(P)@l$ delegates a task P to a node l by spawning a process in node l.

Example 2: Road apple attack (continued) For a node representing a room, $secureRoom ::_s^\delta nil$, the access control policy δ defines the conditions under which other entities can enter or leave the secure room. The set s contains the names of all nodes that are located in the room or connected to the room. Let an insider and an employee be in a hall $hall ::_{\{insider,\, employee.\, secureRoom\}}^\delta nil$ which is neighboring the secure room. An example of an insider delegating a task to the employee is: $insider ::_s^\delta eval(P)@employee^{insider}$ where P is a process denoting the task, *employee* is the node to which the task is delegated and the label *insider* is the origin node. An employee entering the room as part of the task delegated from an insider is presented through $employee ::_s^\delta login(secureRoom)^{insider}.P'$, while an employee leaving the room $employee ::_s^\delta logout(secureRoom)^{insider}.P''$. This example shows that the actions *login* and *logout* abstract from objects leaving or entering locations. The same actions can be used to specify objects being put into or removed from other objects. To keep the level of abstraction sufficiently high and consistent with the constructs

presented by Bettini et al. [11], the action names are generic rather than named specifically, such as "put/take" or "enter/leave".

An origin node can grant a set of capabilities $C = \{ln, lt, e\}$ to another node, where ln is a capability to execute the action *login*, lt to execute the action *logout* and e to execute the action *eval*. Which capabilities the origin node can grant depends on its identity, location and credentials. The access control policy δ is a function $\delta : (\mathcal{L} \cup \{\bot\}) \times (\mathcal{L} \cup \{\bot\}) \times 2^{\mathcal{L}} \rightarrow 2^{C}$. The first and the second parameter denote identity based access control and location based access control respectively. If the identity or the location does not influence the policy, it is replaced by $\bot$. The third parameter denotes credential based access control, which requires a set of credentials to allow an action. If a policy is not affected by credentials, the third parameter is an empty set. A security policy can present a situation where: 1) only credentials are needed, such as a door that requires a key $(\bot, \bot, \{key\}) \mapsto \{ln\}$, 2) only the identity is required, such as a door that requires biometrics information $(John, \bot, \emptyset) \mapsto \{ln\}$, or 3) only the location is required, such as data that can be reached only locally $(\bot, office, \emptyset) \mapsto \{ln\}$. The policy supports combinations of these attributes, such as a door requiring biometrics and a key $(John, \bot, \{key\}) \mapsto \{ln\}$. The policies focus on the allowed action, not of the content of the action. For example, the policy $(insider, \bot, \emptyset) \rightarrow \{ln\}$, at a node *employee*, states the employee trusts the insider sufficiently to accept any object from her. The least restrictive policy that can be used is: $(\bot, \bot, \emptyset) \mapsto \{ln, lt, e\}$.

$$
\begin{aligned}
&world ::^{(\bot,\bot,\emptyset) \mapsto \{ln,lt\}}_{\{remoteServer,\ insider,\ hall\}}\ nil \\
\|\ &hall ::^{(\bot,\bot,\emptyset) \mapsto \{ln,lt\}}_{\{employee,\ secureRoom\}}\ nil \\
\|\ &secureRoom ::^{(employee,\bot,\emptyset) \mapsto \{ln,lt\}}_{\{secureServer\}}\ nil \\
\|\ &remoteServer ::^{(\bot,\bot,\emptyset) \mapsto \{ln\}}_{\{\}}\ nil \\
\|\ &insider ::^{(\bot,\bot,\emptyset) \mapsto \{ln,lt,e\}}_{\{dongle\}}\ P_1 \\
\|\ &employee ::^{(insider,\bot,\emptyset) \mapsto \{ln\}\ ;\ (employee,\bot,\emptyset) \mapsto \{ln,lt,e\}}_{\{\}}\ P_2 \\
\|\ &secureServer ::^{(\bot,secureRoom,\emptyset) \mapsto \{ln,lt\}\ ;\ (\bot,secureServer,\emptyset) \mapsto \{ln,lt\}}_{\{serverData\}}\ nil \\
\|\ &dongle ::^{(\bot,\bot,\emptyset) \mapsto \{e\}\ ;\ (dongle,\bot,\emptyset) \mapsto \{ln,lt\}}_{\{rootkit\}}\ P_3 \\
\|\ &rootkit ::^{(dongle,\bot,\emptyset) \mapsto \{ln,lt,e\}}_{\{\}}\ P_4 \\
\|\ &serverData ::^{(\bot,secureServer,\emptyset) \mapsto \{e\}}_{\{\}}\ nil
\end{aligned}
$$

Fig. 7. The road apple attack environment in the Portunes language

The three layers of nodes, spatial, object and digital, stratify the nodes and guarantee that no invariants can be violated. We also need to avoid impossible containment relationship between nodes from the same layer, such as a node containing itself. We use types to define ordering among nodes from the same layer for a specific model. Each node has a type $t \in T$, where T is a finite partially

ordered set defined by the relation $\succ\succ_{ln}$. The function $\mathcal{T}$ maps a node to its type $\mathcal{T} : N \to T$. The relation $\succ\succ_{ln}$ provides ordering between nodes based on their type. As a convention, we write types with a capital first letter. For the model from the above example, T is defined as $T = \{Room, Person\}$, and the ordering relation as $\succ\succ_{ln} = \{(Room, Person)\}$. The mapping between the nodes and their types is: $\mathcal{T}(secureRoom) = \mathcal{T}(hall) = Room, \mathcal{T}(employee) = \mathcal{T}(insider) = Person$. The ordering is not transitive: For example, a room can contain a dongle and a dongle can contain digital data. But, the room cannot contain the digital data. Also, the ordering is not reflexive: a dongle might not be able to contain a dongle, nor an insider can contain an employee. The only assumption on $\succ\succ_{ln}$ is that it does not invalidate invariant 7 in Definition 1, or put differently, the relation does not allow the cycles between nodes in the digital layer.

Example 3: Road apple attack (continued) In section 2.2 we introduced the Portunes model of the environment where the road apple attack takes place. We defined the relation between the elements through a graph and their stratification in the graph through the function $\mathcal{D}$. Now, we additionally define the $\succ\succ_{ln}$ relation and the security policies on each of the nodes.

Figure 7 presents the environment as a net composition. The representation contains detailed information about the security policies in place, making it suitable for analysis. Note the policy at employee $(employee, \bot, \emptyset) \to \{ln, lt, e\}$ which states that the employee will accept all actions originating from herself. Removing this policy would prevent the node executing any action using its own privileges.

The available types are $T = \{Space, Person, Server, ItObject, Data\}$. The mapping and the Hasse diagram are given in Figure 8. The ordering relation is:

$$\succ\succ_{ln} = \{(Space, Person), (Space, Server), (Space, ItObject), (Person, ItObject), (Server, ItObject), (Person, Server), (ItObject, Data), (Server, Data)\}$$

$\mathcal{T}(world) = \mathcal{T}(hall) = \mathcal{T}(secureRoom) = Space,$
$\mathcal{T}(employee) = \mathcal{T}(insider) = Person,$
$\mathcal{T}(remoteServer) = \mathcal{T}(secureServer) = Server,$
$\mathcal{T}(rootkit) = \mathcal{T}(serverData) = Data,$
$\mathcal{T}(dongle) = ItObject.$

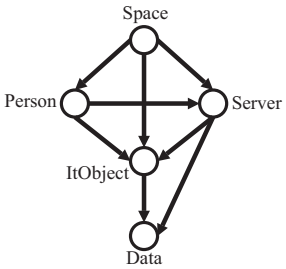

Fig. 8. Type definition of the nodes and the Hasse diagram

A net N is a formal representation of the Portunes model. A net N, together with the mapping functions $\mathcal{D}$, $\succ\succ_{ln}$ on its nodes presents a single state of the environment. The processes P_1 to P_4 represent intentions of the nodes insider, employee, dongle and rootkit respectively.

3.3 Auxiliary functions

Having defined the behavior of nodes using the three primitive actions, *login*, *logout* and *eval*, we now look at the context where these actions can be executed. A node $l ::^{\delta}_{s} a^{l'}.P$ can be restricted in executing an action a from an origin node l' to a target node for four reasons: (1) the origin node might not have sufficient privileges, (2) execution of an action invalidates the invariants in Definition 1, (3) the target node might not be in the vicinity of the node l or (4) the target node is not physically able to contain the node. This section defines the auxiliary functions for a given net N, which take care of these restrictions. The auxiliary functions are defined in Figure 9 and are used in the operational semantics of the language.

$$grant(l_o, \delta_t, a) = \exists k_1, k_2 \in \mathcal{L} \cup \{\bot\}, \exists K \in \mathcal{P}(\mathcal{L}) : a \in \delta_t(k_1, k_2, K) \wedge$$
$$\underbrace{(k_1 = l_o \vee k_1 = \bot)}_{(1)} \wedge \underbrace{(k_2 \in parents_N(l_o) \vee k_2 = \bot)}_{(2)} \wedge \underbrace{(K \subseteq children_N(l_o))}_{(3)},$$
$$\text{where } parents_N(l_o) = \{\, l_{po} |\, l_{po} ::^{\delta_{po}}_{s_{po}} R \in N \wedge\ l_o \in s_{po}\} \text{ and}$$
$$children_N(l_o) = \{\, s_o |\, l_o ::^{\delta_o}_{s_o} R \in N\}$$

$$l_t \succ_{ln} l = \begin{cases} false & \text{iff } (\mathcal{D}(l_t){=}D \wedge (\mathcal{D}(l){=}O \vee \mathcal{D}(l){=}S) \vee (\mathcal{D}(l_t){=}O \wedge \mathcal{D}(l){=}S) \\ & \quad \vee (\mathcal{D}(l_t){=}S \wedge \mathcal{D}(l){=}D) \\ \mathcal{T}(l_t) \succ\!\succ_{ln} \mathcal{T}(l) & otherwise \end{cases}$$

$$l \succ_e l_t = \underbrace{(\mathcal{D}(l) \neq L \wedge \mathcal{D}(l_t) \neq L)}_{(4)} \wedge \underbrace{\neg(\mathcal{D}(l) = D \wedge \mathcal{D}(l_t) = O)}_{(5)}$$
$$\wedge\, (\underbrace{l_t \in children_N(l)}_{(6)} \vee (\underbrace{\exists l_p ::^{\delta_p}_{s_p} R \in N : l \in s_p \wedge l_t \in s_p}_{(7)}) \vee \underbrace{\mathcal{D}(l_t) = D}_{(8)})$$

Fig. 9. Auxiliary function *grant* and $\succ$ relations

The *grant* function checks if an origin node l_o has sufficient privileges to execute an action a on a target node with policy δ_t. The first parameter is the name of the origin node l_o, the second parameter is the policies on the target node δ_t and the third parameter is a label of an action a. A node can execute an action depending on the identity l_o of the origin node (1), its location $parents(l_o)$ (2) or the keys $children(l_o)$ it contains (3). Note that the value of *grant* depends solely of the origin node, not the node executing the process.

The relation $l_t \succ_{ln} l$ states that node l_t can contain a node l. The goal of this relation is to ensure the invariants 3-6 in Definition 1 are satisfied during the net evolution. From the relation we see that a digital node cannot contain spatial or physical node, an object node cannot contain a spatial node and a spatial node cannot contain a digital node.

The ordering relation $l \succ_e l_t$ states that node l can delegate a task to node l_t by means of spawning a process. The relation restricts delegation of tasks

between nodes depending on the layer a node belongs to and the proximity between nodes. An object node can delegate a task to a digital node or another object node, while a digital node can delegate a task only to another digital node. Thus, spatial nodes cannot delegate tasks, nor can a task be delegated to spatial nodes (4), and digital nodes cannot delegate tasks to object nodes (5). Furthermore, a non-digital node can delegate a task only to nodes it contains (6) or nodes that are in the same location (7). In digital nodes the proximity does not play any role in restricting the delegation of a task (8). The decision (8) assumes the world is pervasive and two digital nodes can delegate tasks from any location as long as they have the appropriate privileges.

The expressions from Figure 9 focus on the relation between nodes. The *grant* function provides the security constraints in the language based on the location and identity nodes, while the $\succ_{ln}$, $\succ\succ_{ln}$ and $\succ_e$ relations provide non-security constraints derived from the layer the nodes belong to and their location. In addition, we put a restriction on the processes inside a node, to distinguish tasks originating from a single node. We call such processes simple processes, and define an additional auxiliary function *origin*, which helps to determine if a process is a simple process.

Definition 2. *Let* $origin : Proc \rightarrow 2^{\mathcal{L}}$ *be a function which returns all the action labels of a given process.*

$$origin(nil) = \{\}, origin(a^l.P) = \{l\} \cup origin(P)$$

A process P*, which is either nil or which contains actions only from one origin node is a simple process:* $origin(P) \subseteq \{l_0\}$

In the semantics of the Portunes language this function forbids processes from one origin to spawn processes from other origins. For example, the process definition

$insider ::_s^{\delta} (eval(logout(hall)^{employee}.login(secureRoom)^{employee})@insider)^{insider}$

is not allowed. This process definition can be interpreted as: the insider delegates herself a task to enter the secure room using the privileges from the employee. The execution of this process does not require any interaction with the employee and does not represent a realistic scenario. We also found that the "behavior" of processes can be better mapped in real life scenarios if they execute actions only from a single origin. Naturally, a node can still execute processes from other origins in parallel.

3.4 Operational semantics

Following Bettini et al. [11], the semantics of the Portunes language is divided into process semantics and net semantics. The process semantics is given in terms of a labeled transition relation $\xrightarrow{a}$ and describes both the intention of a process to perform an action and the availability of resources in the net. The label a contains the name of the node executing the action, the target node, the origin node and a set of node names which identify which nodes the target node

contains. The net semantics is given in terms of a transition relation $\Rightarrow$ describes possible net evolutions and relies on the labeled transition relation $\xrightarrow{a}$ from the process semantics.

$$\frac{origin(P) \subseteq \{l_o\} \quad l_t \succ_{ln} l \quad grant(l_o, \delta_t, ln)}{l ::_s^\delta login(l_t)^{l_o}.P \| \, l_t ::_{s_t}^{\delta_t} Q \xrightarrow{login(l,l_t,l_o,s_t)} l ::_s^\delta P \| \, l_t ::_{s_t \cup \{l\}}^{\delta_t} Q} \quad \textbf{[login]}$$

$$\frac{origin(P) \subseteq \{l_o\} \quad grant(l_o, \delta_t, lt) \quad l \in s_t}{l ::_s^\delta logout(l_t)^{l_o}.P \| \, l_t ::_{s_t}^{\delta_t} Q \xrightarrow{logout(l,l_t,l_o,s_t)} l ::_s^\delta P \| \, l_t ::_{s_t \setminus \{l\}}^{\delta_t} Q} \quad \textbf{[logout]}$$

$$\frac{origin(P) \subseteq \{l_o\} \quad origin(Q) \subseteq \{l_o\} \quad l \succ_e l_t \quad grant(l_o, \delta_t, e)}{l ::_s^\delta eval(Q)@l_t^{l_o}.P \| \, l_t ::_{s_t}^{\delta_t} R \xrightarrow{eval(l,l_t,l_o,Q)} l ::_s^\delta P \| \, l_t ::_{s_t}^{\delta_t} R|Q} \quad \textbf{[eval]}$$

$$\frac{l ::_s^\delta P \xrightarrow{a} l ::_s^\delta P'}{l ::_s^\delta P|Q \xrightarrow{a} l ::_s^\delta P'|Q} \quad \textbf{[pComp]}$$

Fig. 10. Process semantics

The process semantics of the language is defined in Figure 10. A node can login to another node **[login]** if it has sufficient privileges to perform the action (*grant*), if the node can be contained in the target node ($\succ_{ln}$) and if the process is a simple process with origin node l_o (*origin*). As a result of executing the action, node l enters node l_t, or put differently, the target node l_t now contains node l. For a node to logout from a target node **[logout]**, the target node must contain the node ($l \in s_t$), the origin node must have proper privileges (*grant*) and the process must be a simple process with origin node l_o (*origin*). The action results in l leaving l_t, specified through removing its node name from s_t. Spawning a process **[eval]** requires both the node executing the action and the target node to be close to each other or the target node to be digital ($l \succ_e l_t$), the origin node should have the proper privileges (*grant*) and both processes P and Q need to be simple processes with origin node l_o (*origin*). The action results in delegating a new task Q to the target node, which contains actions originating from the same origin node as the task P. Note that for delegation to occur, in the Portunes language it is sufficient for the employee (delegatee) to trust the insider(delegator), rather than requiring mutual trust between them. The reason behind this design decision is that we are interested whether the insider can convince the employee to execute a task, rather than whether the insider trusts the employee.

The net semantics in Figure 11 uses the process semantics to define the possible actions in the Portunes language. Spawning a process is limited solely by the process semantics **[neteval]**. To move, a node executes the logout and login actions in sequence **[netmove]**. Both actions should have the same origin node and should be executed by the same node. Furthermore, an object node can move only to a node in its vicinity, while digital nodes do not have this restriction ($l_{t_1} \in s_{t_2} \vee l_{t_2} \in s_{t_1} \vee \mathcal{D}(l) = D$). Data can be copied, which is presented by data

$$\frac{N \xrightarrow{eval(l,l_t,l_o,P)} N_1}{N \xRightarrow{neteval(l,P,l_t)} N_1} \textbf{[neteval]} \qquad \frac{N_1 \xRightarrow{a} N_1'}{N_1 \parallel N_2 \xRightarrow{a} N_1' \parallel N_2} \textbf{[nComp]}$$

$$\frac{N \xrightarrow{logout(l,l_{t_1},l_o,s_{t_1})} N_1 \quad N \xrightarrow{login(l,l_{t_2},l_o,s_{t_2})} N_2 \quad \mathcal{D}(l) = D}{N \xRightarrow{netcopy(l,l_{t_1},l_{t_2})} N_2} \textbf{[netcopy]}$$

$$\frac{N \xrightarrow{logout(l,l_{t_1},l_o,s_{t_1})} N_1 \; N_1 \xrightarrow{login(l,l_{t_2},l_o,s_{t_2})} N_2 \;\; (l_{t_1} \in s_{t_2} \vee l_{t_2} \in s_{t_1} \vee \mathcal{D}(l) = D)}{N \xRightarrow{netmove(l,l_{t_1},l_{t_2})} N_2} \textbf{[netmove]}$$

Fig. 11. Net semantics

entering a new node without leaving the previous **[netcopy]**. Although the data can be copied, it still needs permission from both the node it resides at l_{t_1} and from the node it is copied to l_{t_2}. In the rule this is presented by a possibility of the net N evolving into both net $N_1(logout)$ and $N_2(login)$. However, only the transition to the net $N_2(login)$ is applied. The standard rules for structural congruence apply and are presented in Figure 12.

(ProcCom)	$P_1 \mid P_2 \equiv P_2 \mid P_1$
(NetCom)	$N_1 \parallel N_2 \equiv N_2 \parallel N_1$
(Abs)	$P_1 \mid nil \equiv P_1$

Fig. 12. Structural congruence of processes and nets

Definition 3. *Vicinity of a node l with a parent node l_{t_1} is defined by all nodes l_{t_2} that share the same parent node ($l_{t_2} \in s_{t_1}$) or the child of l_{t_2} is a parent of l: ($l_{t_1} \in s_{t_2}$).*

Proposition 1. *Nodes from the object and spatial layer can move only to nodes in their vicinity.*

Proof. (Sketch) Follows from the netmove premise: $l_{t_1} \in s_{t_2} \vee l_{t_2} \in s_{t_1}$.

Proposition 2. *Nodes from the object and spatial layer can evaluate processes only to child and sibling nodes.*

Proof. (Sketch) The property follows from the premise of the eval action: $\succ_e$.

Theorem 2. *Let $(G, \mathcal{D})$ be a Portunes model and N be a net which represents the model using the Portunes language. The function Map maps a net in a Portunes model, such that $C(Map(N), \mathcal{D})$ holds. The evolutions of the net N do not invalidate the invariants C.*

Proof. The proof is presented in the appendix.

Example 4: Road apple attack(continued) In Section 3.2 we formally specified the environment where the road apple attack occurs. By using the language semantics it is now possible to reason about possible attack scenarios. An attack scenario is presented through defining the processes in the nodes, that lead to an undesired state.

$$
\begin{aligned}
&P_1{=}logout(world).login(hall). && \text{(a)}\\
&\quad eval(logout(insider).login(hall).logout(hall).\\
&\quad login(employee))@dongle && \text{(b)}\\
&P_2{=}logout(hall).login(secureRoom).\\
&\quad eval(logout(employee).login(secureRoom).\\
&\quad logout(secureRoom).login(secureServer))@dongle. && \text{(c)}\\
&P_3{=}eval(logout(dongle).login(secureServer))@rootkit\\
&P_4{=}eval(login(remoteServer))@serverData
\end{aligned}
$$

Fig. 13. Process definitions enabling the road apple attack

Figure 13 shows an example of the actual road apple attack as four processes, P_1, P_2, P_3 and P_4. All actions in the process P_1 have an origin node *insider*, in P_2 an origin node *employee*, in P_3 an origin node *dongle* and in P_4 an origin node *rootkit*. For clarity, the labels on the actions representing the origin node are omitted from the process definitions.

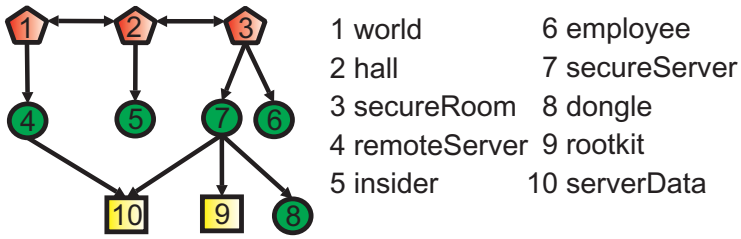

Fig. 14. Portunes model of the road apple attack environment after the execution of the attack

The insider (P_1) goes in the hall and waits for the employee (process P_1 until reaches point a). Then, the insider gives the employee the dongle containing the rootkit, which the employee accepts (P_1 reaches b). Later, the employee plugs the dongle in the secure server (P_2 reaches c) using its own credentials and the server gives the dongle (P_3) access to the local data. When the rootkit (P_4) reaches the server, it copies all the data to the remote server. The above actions represent the road apple attack with a dongle automatically running when attached to a computer [12]. After executing the processes from Figure 13, the data will reside

in the remote server, presented through an edge $(remoteServer, data)$ in the Portunes model in Figure 14.

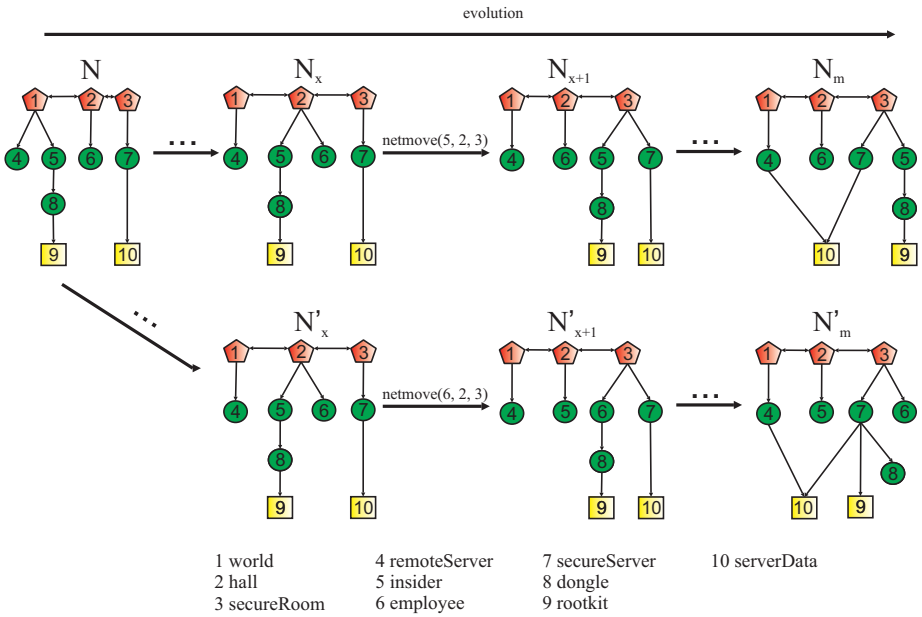

Fig. 15. Example of a net evolution

A net and two possible evolutions are presented in Figure 15. Both of these evolutions lead to the insider obtaining the server data.

Securing the environment from insider threats requires (1) aligning the policies on nodes with the organizational policies and (2) checking if an insider can achieve a specific goal. Examples of such organizational policies are *the server data should never leave the secure server* or *no person except the employee can enter the secure server*. These policies specify the behavior of the whole environment instead of a single node.

First, Portunes can find inconsistencies between organizational and node policies by finding processes that violate an organizational policy. Second, after aligning the node policies with the organizational policies, a user can still check if an insider can achieve a specific goal, without violating any node policies. Expressing organizational policies and formally specifying goals for an insider is the subject of the next section.

4 Expressing properties of Portunes models

In section 3 we defined a language to describe a Portunes model and used the road apple attack as an example. However, we defined the goal of the road apple attack informally, saying that *eventually the data should end up at the remote server*.

The informal presentation of properties in the model is not suitable for stating complex goals because of the ambiguity in informal statements. Furthermore, a formal definition of the desired properties allows a computer aided analysis of a Portunes model.

In this section we define a logic for Portunes in order to (1) describe adversarial goals and (2) define organizational policies which should hold for all evolutions of a Portunes model. The logic is primarily aimed to help penetration testers describe specific adversarial goal and isolate specific subsets of attack scenarios. A logic for Portunes can also aid security auditors to specify organizational policies that should always hold in the organization. Then Portunes can search for scenarios where a person invalidates an organizational policy without invalidating any policies specified on the nodes.

4.1 Motivating examples

The requirements presented in this section are distilled from observing multiple Portunes models obtained through a use case and a series of penetration tests. In the use case, we modeled a five story building and observed the policies on individual objects and the general organizational policies. We also performed a series of physical penetration tests using social engineering [13]. From the attack traces obtained from the tests, we looked at which properties a penetration tester might be interested in. We present our findings in three general requirements.

For each requirement, we provide four motivating examples from which the majority present properties from the road apple attack and are linked to the nodes in Figure 4. The examples are numerated in the form x.y, where x specifies the requirement the example is trying to clarify, and y is the number of the example. The first two examples from each requirement specify properties that are useful for penetration testers, while the second two examples specify properties that are of interest to security auditors.

Requirement 1: *The logic should be able to specify knowledge, location and possession.* We consider that an attack has occurred when unauthorized person eventually (a) learns confidential information, (b) reaches a restricted location or (c) gains possession of an object.

Example 1.1 The server data reaches a remote server.
Example 1.2 The insider learns the employee's password.

Similarly, if we want to specify organizational goals, we can translate the properties as high level policies. These goals should be satisfied for all evolutions of the model.

Example 1.3 The server data should never leave the secure server.
Example 1.4 Only an employee can enter the secure room.

Requirement 2: *The logic should be able to distinguish among multiple evolutions leading to the same goal.* In a penetration test, where the quality of the security is measured by how close the tester gets to the target (the number of circumvented layers of protection), the tester is interested for specific class of attack scenarios.

Example 2.1 The insider enters the secure room and steals the data.
Example 2.2 The insider gives a dongle to the employee and steals the data.

The scenario defined by the process definitions in Figure 13 satisfies the property of example 2.1 and example 2.2 is satisfied by the scenario defined by the process definitions in Figure 16. Both scenarios, eventually achieve the same result. However, a penetration tester might be more interested in scenarios satisfying the first example where the insider as part of the data theft manages to enter the secure room because in these scenarios she circumvents more protection layers and is in that sense these scenarios are better. Therefore, logic should be able to distinguish among multiple evolutions, however it is the user who associates value judgments on the evolutions.

P_1=$logout(world).login(hall).eval(P')@secureServer$
P'=$eval(login(remoteServer)@serverData$
P_2=$eval(logout(hall).login(secureRoom))@insider$
P_3=nil
P_4=nil

Fig. 16. Alternative attack scenario

From a defensive point of view, a security auditor might be interested in specifying the proper execution order of procedures for accomplishing a task.

Example 2.3 A person can enter the secure room only through the hall.
Example 2.4 Whenever the employee receives money, the money is deposited in the secure room.

Requirement 3: *The logic should enable segregation of scenarios based on the social interaction between people, namely trust and delegation.* In Portunes trust is presented through security policies on people, while delegation is described through remote evaluation of processes on people. For example, P_2 in Figure 16, shows that the employee asks the insider to enter the secure room, or in other words *delegates* a task to the insider, which the insider gladly accepts. However, in P_1 in Figure 13, the insider gives the dongle to the employee, and the employee *trusts* the insider sufficiently to accept the dongle.

In some penetration tests the interaction between the tester and an employee is forbidden by the rules of engagement, or it is considered as a risky action because the outcome of the interaction is unpredictable. In other tests the main goal of the tester is to investigate the reaction of the employees when in specific situations. For the first or for the second reason, penetration testers need to isolate attack scenarios that include social interaction.

Example 3.1 The insider steals the data by tricking the employee.
Example 3.2 The insider steals the data without interacting with people.

From defensive point of view, the security auditor might want to check policies on the hierarchy of the organization:

Example 3.3 No person should delegate tasks to the boss.
Example 3.4 Only the boss should delegate tasks to other employees.

		Examples											
Requirement		1.1	1.2	1.3	1.4	2.1	2.2	2.3	2.4	3.1	3.2	3.3	3.4
1	a	y	n	y	y	y	y	y	y	y	y	n	n
	b	n	y	n	n	n	n	n	n	n	n	n	n
	c	n	n	n	n	n	n	n	y	n	n	n	n
2		n	n	y	y	y	y	y	y	y	y	y	y
3		n	n	n	n	n	y	n	n	y	y	y	y

Fig. 17. The requirements and the examples that motivate the requirements.

In Figure 17 we provide an overview of the requirements and show which property the logic should be capable to express to specify each motivating example. For example, expressing the property in Example 3.2 requires the logic to be able to express location, to show that the data is in a server controlled by the insider (requirement 1.a), to segregate among subsets of net evolutions, to select only evolutions where the insider tricks the employee (requirement 2) and to express interactions between people, to show the interaction between the insider and the employee (requirement 3).

4.2 Net and net evolution predicates

Motivated by the examples above, we now present the logic for expressing properties of a Portunes model. First we introduce the syntax of predicates for locations, actions and processes and provide their semantics. Using these predicates we can specify properties on process definitions for a given net. Next, we present the predicates on the transition labels that describe the net evolutions. Finally, we present the semantics of the modal logic used for describing the properties of a given net.

4.3 Net predicates

Syntax of location predicates:

$$lp ::= 1_l \mid u \mid l$$

Location predicates can be a generic location (1_l), a type (u) or a location (l). $\mathcal{L}$ is a universe of node names, $\mathcal{L}_N \subseteq \mathcal{L}$ is a finite set of names for a given net N and Loc_N is a finite set of location predicate atoms for a given net N.

Examples of location predicates are: the predicate *insider* is satisfied by all nodes named *insider*, the predicate *Space* is satisfied by all nodes of type *Space* and 1_l is satisfied by all nodes in the net.

Syntax of action predicates:

$$ap ::= lt(lp) \mid ln(lp) \mid e(pp)@lp$$

Every action from the Portunes language is represented by a predicate. The predicate $lt(lp)$ is satisfied by all $logout(l)$ where l satisfies the location predicate lp. Similarly, the action predicate $ln(lp)$ is satisfied by all $login(l)$ actions where l satisfies the location predicate lp and $e(pp)@lp$ is satisfied by all $eval(P)@l$, where the process P satisfies the predicate pp and the node l satisfies the predicate lp. $\mathcal{A}$ is a universe of actions, $\mathcal{A}_N \subseteq \mathcal{A}$ is a finite set of actions in a given net N and Act_N is a finite set of action predicates for a given net N.

A few examples of action predicates are: the predicate $lt(insider)$ is satisfied by all the actions $logout(insider)$, $ln(Space)$ is satisfied by all *login* actions that perform login to a node of type *Space*, and $e(1_p)@employee$ is satisfied by all *eval* actions that delegate a process to a node that satisfy the predicate *employee*.

Syntax of process predicates:

$$pp ::= 1_p \mid pp \wedge pp \mid ap^{lp} \rightarrow pp$$

The process predicate 1_p is satisfied by all processes and a conjunction of two process predicates $pp \wedge pp$ is satisfied by processes that satisfy both predicates. The predicate $ap^{lp} \rightarrow pp$ is satisfied by processes that contain an action satisfying the action predicate ap with an origin node satisfying the predicate lp followed by a process that satisfies the pp predicate. We define $\mathcal{P}$ as a universe of processes, $\mathcal{P}_N \subseteq \mathcal{P}$ as a finite set of processes in a given net N and $Proc_N$ as a finite set of process predicates for a given net N.

For example, the net:

$$N ::= insider ::^{\delta}_{\{money\}} P \mid\mid employee ::^{\delta}_{\{secret\}} Q \mid\mid hall ::^{\delta}_{\{insider,employee\}}$$

defines an environment where an employee and an insider are in the same hall. The intention of the insider to give money to the employee can be presented through the process predicate:

$$(e(ln(employee)^{insider} \rightarrow 1_p)@money^{insider}) \rightarrow 1_p.$$

The process predicate has the form $ap^{lp} \rightarrow pp$, where the action predicate ap is $e(ln(employee)^{insider} \rightarrow 1_p)@money$, the origin predicate of ap is $insider$ and pp is the predicate 1_p. The action predicate ap is of the form $e(pp)@lp$ where the process predicate is again of the from $ap^{lp} \rightarrow pp$, or $e(ln(employee)^{insider} \rightarrow 1_p$ and

the locality predicate of the form *money*. This process predicate will be satisfied by all processes originating from *insider* that contain an action $eval(P)@money$. Moreover the process P must contain an action $login(employee)$ originating from *insider*. Similarly the intention of the employee to give a secret to the insider is presented through the predicate:

$$(e(ln(insider)^{employee} \rightarrow 1_p)@secret^{employee}) \rightarrow 1_p.$$

Two processes that satisfy these predicates are:

$$P = _{eval(logout(insider)...login(employee))@money^{insider}}$$
$$Q = _{eval(logout(employee)...login(insider))@secret^{employee}}$$

4.4 Semantics of state predicates

Having defined the syntax of the predicates, now we present the semantics in the form of the functions: $\mathbb{L} : Loc_N \rightarrow 2^{\mathcal{L}_N}, \mathbb{AC}: Act_N \rightarrow 2^{\mathcal{A}_N}, \mathbb{P} : Proc_N \rightarrow 2^{\mathcal{P}_N}$, which take a predicate lp, pp or ap and return a set of locations, processes and actions that satisfy the predicates respectively. The sets $\mathcal{L}_N$, $\mathcal{A}_N$ and $\mathcal{P}_N$ are derived from a named Portunes model presented by a specific net N. The semantics are defined in Figure 18.

$$\begin{array}{ll}
\mathbb{L}\,[\![1_l]\!] = \mathcal{L}_N & \mathbb{AC}[\![lt(lp)\,]\!] = \{logout(l) \mid \exists l : l \in \mathbb{L}\,[\![lp]\!]\} \\
\mathbb{L}\,[\![u\,]\!] = \{l \mid \mathcal{T}(l) = u\} & \mathbb{AC}[\![ln(lp)]\!] = \{login(l) \mid \exists l : l \in \mathbb{L}\,[\![lp]\!]\} \\
\mathbb{L}\,[\![\,l\,]\!] = \{l\} & \mathbb{AC}[\![e(pp)@lp]\!] = \{eval(P)@l \mid \exists l, P : l \in \mathbb{L}\,[\![lp]\!], P \in \mathbb{P}[\![pp]\!]\}
\end{array}$$

$$\begin{array}{l}
\mathbb{P}\,[\![1_p]\!] = \mathcal{P}_N \\
\mathbb{P}\,[\![pp_1 \wedge pp_2]\!] = \mathbb{P}[\![pp_1]\!] \cap \mathbb{P}[\![pp_2]\!] \\
\mathbb{P}\,[\![ap^{lp} \rightarrow pp]\!] = \{P \mid \exists a,l,Q : a \in \mathbb{AC}[\![ap]\!], l \in \mathbb{L}\,[\![lp]\!], origin(a) = l, P \xrightarrow{a}{}^{+} Q, Q \in \mathbb{P}[\![pp]\!]\}
\end{array}$$

Fig. 18. Interpretation of location, action and process predicates

The relation $P \xrightarrow{a}{}^{+} Q$ is satisfied when: $\exists P' : P \rightarrow^* P', P' \xrightarrow{a} Q$, where $\rightarrow^*$ is the reflexive, transitive closure of $\rightarrow$.

$\mathbb{L}\,[\![1_l]\!]$ returns the set of locations $\mathcal{L}_N$, $\mathbb{L}\,[\![u]\!]$ returns a set of locations that belong to a specific type and $\mathbb{L}\,[\![l]\!]$ returns a specific location $l \in \mathcal{L}_N$. $\mathbb{P}[\![1_p]\!]$ returns all processes in the net and $\mathbb{P}[\![pp_1 \wedge pp_2]\!]$ returns the processes that satisfy both predicates pp_1 and pp_2. $\mathbb{P}[\![ap^{lp} \rightarrow pp]\!]$ returns the processes that can execute an action satisfying the predicate ap, using an origin satisfying the predicate lp, and then evolve in a process that satisfies the predicate pp.

A process P from a net N satisfies the predicate pp, iff $P \in \mathbb{P}[\![pp]\!]$. Analogously, action a from a net N satisfies the predicate ap iff $a \in \mathbb{AC}[\![ap]\!]$ and a location l from a net N satisfies the predicate lp iff $l \in \mathbb{L}[\![lp]\!]$.

4.5 Transition label predicates

The process predicates present a set of actions that a single process might perform, and not actual net evolutions. In other words, a process predicate specifies an intention not an execution. The transition labels, which present evolutions of a net are defined in Figure 19.

$$
\begin{array}{rl}
lab ::= & netmove(l,l,l) \\
| & neteval(l,P,l) \\
| & netcopy(l,l,l)
\end{array}
\qquad\qquad
\begin{array}{rl}
A ::= & \circ \\
| & A \cup A \\
| & A \cap A \\
| & A - A \\
| & src(lp) \\
| & trg(lp) \\
| & prt(lp) \\
| & nm(lp_1, lp_2, lp_3) \\
| & ne(lp_1, pp, lp_2) \\
| & nc(lp_1, lp_2, lp_3)
\end{array}
$$

Fig. 19. Syntax of transition labels and transition label predicates

We define the syntax and semantics of label predicates, where the locations and processes are replaced by location and process predicates. We use $\circ$ to denote all transition labels and $\cup$, $\cap$ and $-$ to denote union, intersection and exclusion of two sets of transition labels. The predicates *src*, *prt* and *trg* denote transition labels which have a specific source, parent or target node. The predicate $nm(lp_1, lp_2, lp_3)$ denotes transitions labeled *netcopy* where the first parameter of the transition label satisfies the location predicate lp_1, the second parameter lp_2 and the third parameter lp_3. Similarly, *ne* and *nc* denote the *neteval* and *netcopy* transition labels respectively.

$$
\begin{array}{ll}
\mathbb{A}[\![\ \circ \ ]\!] & = Lab_N \\
\mathbb{A}[\![A_1 \cup A_2]\!] & = \mathbb{A}[\![A_1]\!] \cup \mathbb{A}[\![A_1]\!] \\
\mathbb{A}[\![A_1 \cap A_2]\!] & = \mathbb{A}[\![A_1]\!] \cap \mathbb{A}[\![A_1]\!] \\
\mathbb{A}[\![A_1 - A_2]\!] & = \{a \mid a \in \mathbb{A}[\![A_1]\!], a \notin \mathbb{A}[\![A_2]\!]\} \\
\mathbb{A}[\![\ src(lp) \ ]\!] & = \{a \mid a \in \mathbb{A}[\![nm(l, 1_l, 1_l) \cup nc(l, 1_l, 1_l) \cup ne(l, 1_p, 1_l)]\!], l \in \mathbb{L}[\![lp]\!]\} \\
\mathbb{A}[\![\ trg(lp) \ ]\!] & = \{a \mid a \in \mathbb{A}[\![nm(1_l, 1_l, l) \cup nc(1_l, 1_l, l) \cup ne(1_l, 1_p, l)]\!], l \in \mathbb{L}[\![lp]\!]\} \\
\mathbb{A}[\![\ prt(lp) \ ]\!] & = \{a \mid a \in \mathbb{A}[\![nm(1_l, l, 1_l) \cup nc(1_l, l, 1_l)]\!], l \in \mathbb{L}[\![lp]\!]\} \\
\mathbb{A}[\![nm(lp_1, lp_2, lp_3)]\!] & = \{netmove(l_1, l_2, l_3) \mid l_1 \in \mathbb{L}[\![lp_1]\!], l_2 \in \mathbb{L}[\![lp_2]\!], l_3 \in \mathbb{L}[\![lp_3]\!]\} \\
\mathbb{A}[\![ne(lp_1, pp, lp_2) \ ]\!] & = \{neteval(l_1, P, l_3) \mid l_1 \in \mathbb{L}[\![lp_1]\!], l_2 \in \mathbb{L}[\![lp_2]\!], P \in \mathbb{P}[\![pp]\!]\} \\
\mathbb{A}[\![nc(lp_1, lp_2, lp_3) \ ]\!] & = \{netcopy(l_1, l_2, l_3) \mid l_1 \in \mathbb{L}[\![lp_1]\!], l_2 \in \mathbb{L}[\![lp_2]\!], l_3 \in \mathbb{L}[\![lp_3]\!]\}
\end{array}
$$

Fig. 20. Semantics of transition label predicates

We use Lab_N to denote a finite set of label predicates defined over a given net N and $\mathcal{LP}_N$ to denote a finite set of transition labels. The function that defines the meaning of the label predicates $\mathbb{A} : Lab_N \rightarrow 2^{\mathcal{LP}_N}$ is given in Figure 20.

For example, the predicate $prt(insider)$ is satisfied by all transition labels which add or remove an object or data from the node satisfying the location predicate $insider$, $trg(employee)$ is satisfied by all transition labels in which an object or data is given, or a task is delegated to a node satisfying the location predicate $employee$ and $nm(Person, 1_l, secureRoom) - nm(employee, 1_l, secureRoom)$ is satisfied by all transition labels in which node of type $Person$ other than the node satisfying the predicate $employee$ move to a node that satisfies the predicate $secureRoom$.

4.6 Logic for Portunes models

Definition 4. *(Hennessy-Milner Logic) The set of HML formulas [14] for Portunes is given by the BNF grammar:*

$$\phi ::= tt \mid \neg\phi \mid \phi \wedge \phi \mid c(lp, lp) \mid \langle A \rangle \phi$$

The formula tt is always satisfied. The formula $\neg\phi$ is satisfied by a net that does not satisfy ϕ, while $\phi_1 \wedge \phi_2$ is satisfied by a net that satisfies both ϕ_1 and ϕ_2. The formula $c(lp_1, lp_2)$ is satisfied by a net in which a node with a name satisfying the predicate lp_2 belongs to the set s of a node satisfying the predicate lp_1. Finally, $\langle A \rangle \phi$ is satisfied by a net that satisfies the formula ϕ and is a result of a transition with a label that satisfies A. Formulas like $[A]\phi$ and $\phi_1 \vee \phi_2$ can be derived from the logic: $[A]\phi = \neg\langle A \rangle \neg\phi$ and $\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)$.

For example, $c(remoteServer, serverData)$ is satisfied by all nets where the server data is located in the remote server and $\langle nm(insider, hall, secureRoom) \rangle tt$ is satisfied by all nets where the insider moves from the hall to the secure room. We provide more examples of he formulas in the following section.

Let Net be a set of all nodes for a given network N, and Φ the set of all the formulas. The semantics is defined using the function $\mathbb{M} : \Phi \rightarrow 2^{Net}$:

$$\begin{array}{l}
\mathbb{M}[\![tt]\!] = Net \\
\mathbb{M}[\![\neg\phi]\!] = Net - \mathbb{M}[\![\phi]\!] \\
\mathbb{M}[\![\phi_1 \wedge \phi_2]\!] = \mathbb{M}[\![\phi_1]\!] \cap \mathbb{M}[\![\phi_2]\!] \\
\mathbb{M}[\![\langle A \rangle \phi]\!] = \{N \mid \exists a, N_1 : N \stackrel{a}{\Longrightarrow}^{+} N_1, a \in \mathbb{A}\,[\![A]\!], N_1 \in \mathbb{M}[\![\phi]\!]\} \\
\mathbb{M}[\![c(lp_1, lp_2)]\!] = \{N \mid \exists l_1, l_2 : l_2 \in children_N(l_1), l_1 \in \mathbb{L}\,(lp_1), l_2 \in \mathbb{L}\,(lp_2)\}
\end{array}$$

Fig. 21. Semantics of the logic

A net N satisfies a formula ϕ if and only if $N \in \mathbb{M}\,[\![\phi]\!]$, and we write $N \models \phi$. We write $N \stackrel{a}{\Longrightarrow}^{+} N_1$ iff $\exists N' : N \Longrightarrow^{*} N', N' \stackrel{a}{\Longrightarrow} N_1$. where $\Longrightarrow^{*}$ is a reflexive, transitive closure of $\Longrightarrow$.

The above formulas allow us to specify properties of the net for a single state, for all states in which the net evaluates and properties on the net evolutions.

4.7 Examples revisited

In section 3.2 we used the Portunes language to describe the road apple attack formally, an attack where the adversary uses physical, social and digital means to gain possession of sensitive data. We use the road apple and the examples from the previous section to (1) describe adversarial goals and (2) formally define high level policies which should hold for all evolutions of the net.

Example 1.1 The server data reaches a remote server.

$$\langle\circ\rangle c(remoteServer, serverData)$$

Example 1.2 The insider learns the employees password.

$$\langle\circ\rangle c(insider, employeePassword)$$

In Example 1.1 and 1.2 the goal is defined by a node being at a specific location. Using similar logic constructs, we can express goals including knowledge of information (person contains data) and possession (when person contains object). Note that in the above examples we are not interested in the initial state of the net, but in an eventual state in the future.

Example 1.3 The server data should never leave the secure server.

$$\neg\langle nm(serverData, secureServer, 1_l)\rangle tt \wedge \neg\langle nc(serverData, secureServer, 1_l)\rangle tt$$

Example 1.4 Only an employee can enter the secure room.

$$\neg\langle nm(Person, 1_l, secureRoom) - nm(employee, 1_l, secureRoom)\rangle tt$$

Examples 1.3 and 1.4 describe organizational policies which should never be invalidated. Here we also see how location (similarly knowledge and possession) can be used to define an organizational policy.

Example 2.1 The insider steals the data by entering the secure room.

$$\langle nm(insider, 1_l, secureRoom)\rangle(\langle\circ\rangle c(remoteServer, serverData))$$

Example 2.2 The insider steals the data by giving the employee a dongle.

$$\langle nm(dongle, 1_l, employee)\rangle(\langle\circ\rangle c(remoteServer, serverData))$$

In the above two examples, we define two strategies how the insider might get access to the data. Both properties might be satisfied by a single net evolution. For example, the insider enters the office and then gives the dongle to the employee, or vice versa. Adding additional desired or non-desired conditions further segregates the possible evolutions of the net, allowing the penetration tester to focus only on those evolutions she is interested in.

Example 2.3 A person can enter the secure room only through the hall.

$$\neg\langle nm(Person, Space, secureRoom) - nm(Person, hall, secureRoom)\rangle tt$$

Example 2.4 Whenever the employee receives money, the money is deposited in the secure room.

$$\langle nm(money, 1_l, employee)\rangle(\langle nm(money, employee, secureRoom)\rangle tt)$$

In the examples 2.3 and 2.4, the transition label predicates are satisfied only by a specific subset of the transition labels. Namely, all locations from where an employee can move inside the room, except the hall are forbidden. Or, as is the case of example 2.4, the property specifies only net evolutions where an employee receives money and then the money is eventually sent to the secure room.

Example 3.1 The insider steals the data by tricking the employee.

$$\langle ne(insider, 1_p, employee)\rangle(\langle\circ\rangle c(remoteServer, serverData))$$

Example 3.2 The insider steals the data without interacting with people.

$$\neg\langle ne(insider, 1_p, Person)\rangle(\langle\circ\rangle c(remoteServer, serverData))$$

In some penetration tests, the rules of engagement forbid any interaction with the employees. In other tests, the main goal is to see the resilience of the employees against social engineering. Examples 3.1 and 3.2 show how we can segregate attack scenarios that include contact with a specific person, or contain no contact with people.

Example 3.3 No person should delegate tasks to the boss.

$$\neg\langle ne(Person, 1_p, boss)\rangle tt$$

Example 3.4 Only the boss should delegate tasks to other employees.

$$\neg\langle ne(Person, 1_p, Person) - ne(boss, 1_p, Person)\rangle tt$$

In example 3.3 and 3.4 we show how the social aspects of the Portunes model can be used as organizational policies. Finding a delegation from an employee to a boss, or from an employee to another employee would mean that there is inconsistency in the policies imposed on the employees with the organizational policies.

The examples 1.1, 1.2, 2.1, 2.2, 2.4 and 3.1 present desirable properties on the environment, by defining a set of desirable a) transitions $\langle A\rangle tt$, b) states $\langle\circ\rangle c(lp, lp)$ or c) transitions and states $\langle A\rangle(\langle\circ\rangle c(lp, lp)$ that should occur in a net evolution. Similarly, by adding a negation in front of the formulas we can specify undesirable properties (examples: 1.3, 1.4, 2.3, 3.2, 3.3, 3.4). One can imagine properties which include both, desired and undesired states and transitions.

5 Implementation

The Portunes framework can be encoded in most of the model checking tools, such as NuSMV [15] and SPIN [16]. We implemented a proof of concept of the framework in the Groove model checker [17]. Groove is a tool designed for modeling object-oriented systems using graph transformations as a basis for model transformation and operational semantics. We chose Groove as a model checker because (1) encoding the Portunes framework in a graph is natural, (2) Groove offers an easy and modifiable way to define the semantics, (3) the generated scenarios are easy to visualize and simulate and (4) Groove is open-source, allowing the implementation of custom heuristics in the state exploration strategies. Encoding Portunes in other model checking tools might improve the performance of the framework and we consider it as future work.

We first developed a mapping to translate a net into a Groove state graph, then applied the semantics of the Portunes language through Groove rules and finally used part of the logic presented in Section 4 to describe state and evolution properties.

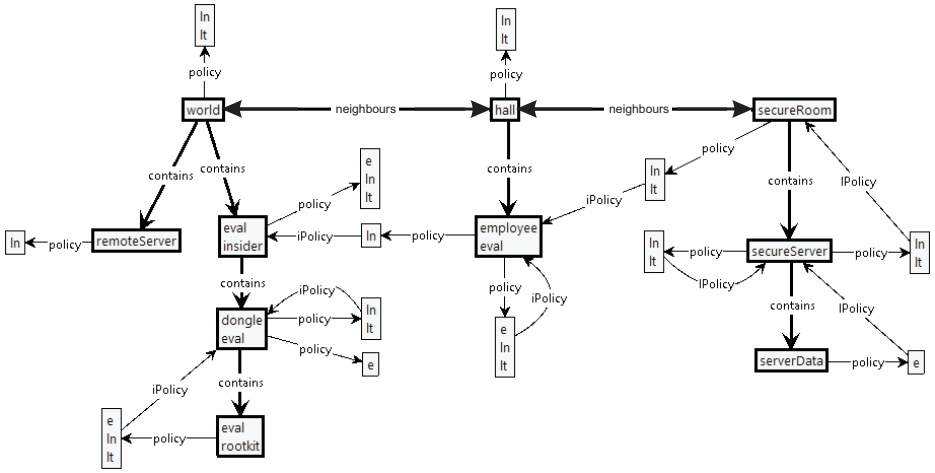

Fig. 22. The road apple in Groove

Translating a net into initial state. To present the net in a graph, we mapped the nodes, the policies and the types u into an attributed graph. The road apple example from Figure 7 is presented (without types of the nodes) using Groove in Figure 22.

Since all the information in Groove is presented as an attributed graphs, we encoded the policies as nodes. Figure 23 provides an example how the policies on *employee* are implemented in Groove:

$$(insider, \bot, \emptyset) \mapsto \{ln\}\ ;\ (employee, \bot, \emptyset) \mapsto \{ln, lt, e\}$$

. The two edges labeled *policy* from *employee* to the set of capabilities $\{ln, lt, e\}$ and $\{ln\}$ define two policies. If the policy grants a capability based on the identity of the origin node, then an *iPolicy* edge points to the identity that is required. Similarly, policies that require the origin node to be at a specific location are presented with *lPolicy* edges and policies that require the origin node to have specific credentials are presented using *cPolicy* edges.

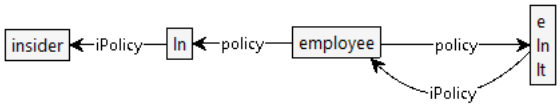

Fig. 23. An example of two policies using Groove

Encoding the Portunes semantics. The net semantics from Figure 11 are presented in Groove as graph transformation rules. The premises of a rule are presented in Groove as a left side graph, while the results of applying the rule as a right side graph. Whenever Groove finds a state which contains the left side graph as a subgraph, it applies a transition and replaces the left side graph with the right side graph.

Expressing the logic in Groove. Groove is capable of expressing first-order logic [18], including the propositions in our logic. State properties in Groove are implemented through rules. For example, the formula

$$c(world, remoteServer) \wedge c(remoteServer, serverData)$$

can be presented as a rule that describes a sub-graph in the state graph (Figure 24). During the state exploration, Groove checks if a state contains the sub-graph, and thus satisfies the property presented through the logic.

Groove is also able to model-check CTL formulas [19] which is a more expressive logic than the modal logic on the net evolutions presented in Section 4. However, in the current proof of concept implementation, we did not define the process definitions. Adding process definitions on nodes requires modification of the code in Groove and we consider it as a future work.

Fig. 24. Expressing properties in Groove

We successfully executed the road apple example in Groove. The complete state space contained 600 states and 2796 transitions. To reduce the number of states we used in the implementation the monotonicity assumption introduced by Ammann et al. [20]. The assumption states that the premise of a given rule

is never invalidated by the successful application of another rule. In the physical world, this assumption means a person able to enter a room can never lose this ability, presenting the most pessimistic scenario where the insider never loses a capability. In the Portunes language, the assumption implies that a node never loses a node it contains, thus in every node $l ::_s^\delta P$ elements can be only added to the set s. Since **[netmove]** removes an element from s, it invalidates the monotonicity assumption. Thus, in the implementation we identified the **[netmove]** rule with the **[netcopy]** rule.

As a result of the monotonicity assumption, it is possible to specify rules in Groove, that only add new edges in the state graph. In a graph of N nodes, there are maximum of N^2 edges possible, and since we do not create any new nodes, the state exploration finishes in maximum $O(N^2)$ steps.

The final state from the exploration contains information about all possible movements of the nodes. Figure 25 presents the final state from the road apple example, where for example, the sever data can end up at three different objects.

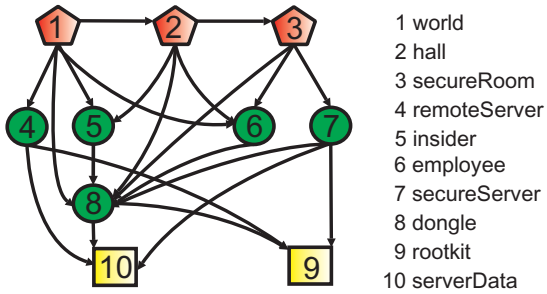

Fig. 25. The final state of the road apple example

Ammann et al. [20] also show that the order of applying the transitions is not important. Because the monotonicity assumption states that the precondition of a given transition is never invalidated by another transition, when there are multiple transitions possible from a specific state, it is not important which transition is executed first.

In linear exploration, when there are multiple transitions from one state, only one random transition is applied. The process continues until no more transitions are possible. Using the monotonicity assumption in this exploration strategy significantly reduces the state space. In the road apple example the state space was reduced to 16 states and 15 transitions.

As a next step, we are interested in (1) using Portunes for generating attack scenarios for physical penetration tests where the testers are allowed to use physical access and social engineering to obtain a digital asset and (2) the usability of the implementation. Previously we developed methodologies for physical penetration testing using social engineering [13] and used the methodologies in orchestrating 15 penetration tests [21].

Currently we are orchestrating 20 additional penetration tests using the same methodologies, where half of the penetration teams use the Portunes framework to generate the attack scenarios and the other half use brainstorm for scenarios. All the models generated by the penetration testers proved to be smaller than 50 nodes, for which Groove needed less then 10 seconds to search the state space using linear exploration. The implementation was able to produce realistic scenarios that the penetration testers actually executed. From the study we hope to obtain profound knowledge on the classes of attacks the framework is suitable for presenting and with which constructs the framework should be extended. After the tests we will use the experience of the penetration testers with the tool to perform a study on the usability of the implementation. We are currently analyzing the data and the results will be presented in a future work.

6 Related work

The design of the Portunes model and Portunes language is influenced by several research directions, such as insider threat modeling, physical modeling and process calculi. This section lists several papers which influenced the design of Portunes and describes how Portunes extends or deviates from them.

Dragovic et al. [22] are concerned with modeling the physical and digital domain to determine data exposure. Their model defines a containment relation between layers of protection. Data security is determined not by access control policies, but by the number of layers of protection above the data and the confidentiality provided by each layer. The Portunes model uses a similar relation to present the location of elements, but uses access control policies to describe security mechanisms. Scott [23] focuses on the mobility of software-agents in a spatial area and usage policies that define the behavior of the agents depending on the locality of the hosting device. The mobility of the agents is restricted through edges on a graph. The Portunes model adds semantics to the graph structure by giving meaning to the nodes and edges and defines invariants enforced directly into the semantics of the language. Mathew et al. [24] use capability acquisition graphs to describe the physical structure of a building. The nodes in the graphs are static, and the graph can present the progress of the insider in the graph. In our solution the structure of the model evaluates as the attack progresses, and the insider can interact with other employees in order to obtain additional capabilities.

Klaim [10] is a process calculus for agent interaction and mobility, consisting of three layers: nodes, processes and actions. There are several Klaim dialects, including μKlaim [25], OpenKlaim [11] and acKlaim [26]. The goal of the acKlaim language, which is closest to our work, is to present insider threats by combining the physical and digital security domain. Mobility is presented by remote evaluation of processes. The Portunes language builds upon these Klaim dialects. Firstly, the actions for mobility and embedding of objects (login, logout) are similar to OpenKlaim. Secondly, the security policies expressed in the Portunes language are similar to acKlaim and μKlaim. However, in the Portunes language

mobility is represented by moving nodes rather than evaluating processes. Finally, the Portunes language lacks tuple spaces which are present in all other Klaim variants, because their meaning in the physical world is completely replaced by the containment set s in the nodes. The absence of tuple spaces reduces the number of possible process definitions, allowing their automatic generation.

The modal logic to reason about properties of a Portunes model and formally present goals of attack scenarios is inspired by the logic for mobile agents of De Nicola et al. [27]. The logic for mobile agents allows specification of mobile system properties specified in Klaim, or more specifically, the tuples residing at specific nodes and the actions that a system performs during its evolution. Our approach uses similar notation and constructs, but adapts the semantics of the logic to the constructs of the Portunes language. First, the Portunes language does not have variables nor logical localities, but does have node types. Thus, the predicates for variables are absent, we do not distinguish between physical and logical localities, and we introduce the type predicate u. Second, the Klaim language has a different set of actions, thus the predicates for Klaim actions are replaced with predicates that reflect Portunes actions. Finally, the Portunes language does not have tuples, thus all tuple predicates from the transition labels and transition label predicates are absent. Because Portunes does not use variables nor logical spatialities all binding constructs and mapping are also absent from the logic.

The implementation of the Portunes framework produces scenarios that invalidate a security goal or an organizational policy. These scenarios are described graphically similarly to the scenario graphs proposed by Wing [28]. Because the attack scenarios can be presented as attack graphs, the same analysis can be applied used in the attack graph community, such as ranking of the scenarios [29], identifying critical assets in the model [30], cost-effective protection strategies from specific attacks [31] and providing quantitative measurement on the overall security of the environment [32]. The main difference of the scenarios produced by Portunes and the scenarios produces by other models is that the Portunes scenarios contain an additional information about the physical layout of the environment and possible interactions between people, rather than information only on the computer network.

7 Conclusion and Future work

The main contribution of this paper is the mapping of security aspects of the physical and social domain together with the digital domain into a single framework named Portunes. This approach allows generating and analyzing attack scenarios which span all tree domains, and thus helps in the protection against insider threat.

The framework consists of a high-level model and a language inspired by the Klaim family of languages. To capture the three domains efficiently, Portunes is able to represent 1) physical properties of elements, 2) mobility of objects and data, 3) identity, credential and location based access control and 4) trust

and delegation between people. In this work we also present a logic to express state and transition properties of the Portunes language. We implemented the framework in an existing model checking tool to generate scenarios where the policies on the individual objects are not violated, but still lead to a security bridge.

We aimed at abstraction level of the three domains to be sufficiently high to be easy to use, but still sufficiently detailed to provide useful results. One can envision extending the framework with constructs such as negotiation between people, behavioral patterns or detection mechanisms, to increase the detail of the produced attack scenarios. We bring the language close to practitioners, by providing an abstract model that hides the details of the language from the user, a logic to help users specify their goals and a graphical implementation of the language in a model checker. This proof of concept implementation can be further developed to suit the needs of penetration testing teams and security auditors.

The applicability of Portunes is demonstrated using the example of the road apple attack, showing how an insider can attack without violating existing security policies by combining actions from all three domains. Currently we are using Portunes to automatically generate attack scenarios for penetration testing teams that use physical access and social engineering to gain possession of a digital asset. So far, we found out that Portunes can produce sufficiently detailed realistic attack scenarios for testers to execute. Future results of this study will provide an insight on the types of scenarios Portunes can produce as well as the usability of the implementation.

References

1. INFOSEC Research Council. Hard problem list. `www.cyber.st.dhs.gov/docs/IRC_Hard_Problem_List.pdf`, 2005.
2. S. Stasiukonis. Social engineering the usb way. `www.darkreading.com/document.asp?doc_id=95556`, 2006.
3. I. Ray and N. Poolsapassit. Using attack trees to identify malicious attacks from authorized insiders. In *Preceedings of ESORICS'05*, pages 231–246. Springer, 2005.
4. E. Van Den Berg, S. Uphadyaya, P.H. Ngo, M. Muthukrishnan, and R. Palan. Mitigating the insider threat using high-dimensional search and modeling. 2006.
5. D. Ha, S. Upadhyaya, H. Ngo, S. Pramanik, R. Chinchani, and S. Mathew. Insider threat analysis using information-centric modeling. In *IFIP International Conference on Digital Forensics*, pages 55–73. Springer, 2007.
6. L. Wang and S. Jajodia. An approach to preventing, correlating, and predicting multi-step network attacks. *Intrusion Detection Systems*, pages 93–128, 2008.
7. M.R. Randazzo, M. Keeney, E. Kowalski, D. Cappelli, and A. Moore. Insider threat study: Illicit cyber activity in the banking and finance sector. *U.S. Secret Service and CERT Coordination Center Software Engineering Institute*, pages 1–25, 2004.
8. J. DePoy, J. Phelan, P. Sholander, B.J. Smith, G.B. Varnado, G.D. Wyss, J. Darby, and A. Walter. Critical infrastructure systems of systems assessment methodology. Technical Report SAND2006-6399, Sandia National Laboratories, 2007.
9. T. Dimkov, Q. Tang, and P. H. Hartel. On the inability of existing security models to cope with data mobility in dynamic organizations. In *Proceedings of the Workshop on Modeling Security*, pages 1–13, 2008. CEUR Workshop Proceedings.
10. R. De Nicola, G. L. Ferrari, and R. Pugliese. KLAIM: A kernel language for agents interaction and mobility. *IEEE Transactions on software engineering*, 24(5):315–330, 1998.
11. L. Bettini, M. Loreti, and R. Pugliese. An infrastructure language for open nets. In *SAC '02: Proceedings of the 2002 ACM Symposium on Applied Computing*, pages 373–377. ACM, 2002.
12. M. AlZarouni. The reality of risks from consented use of usb devices. In C. Valli and A. Woodward, editors, *Proceedings of the 4th Australian Information Security Conference*, pages 5–15, 2006.
13. T. Dimkov, W. Pieters, and P. Hartel. Two methodologies for physical penetration testing using social engineering. In *ACSAC'10*, Chicago, USA, 2010. ACM.
14. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of ACM*, 32(1):137–161, 1985.
15. A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. Nusmv: A new symbolic model verifier. In *Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 682–682. Springer, 1999.
16. G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 2002.
17. A. Rensink. The groove simulator: A tool for state space generation. In J. L. Pfaltz, M. Nagl, and B. Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, volume 3062 of *Lecture Notes in Computer Science*, pages 479–485, Berlin, 2004. Springer Verlag.
18. A. Rensink. Representing first-order logic using graphs. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *International Conference on Graph Transformations (ICGT)*, volume 3256 of *Lecture Notes in Computer Science*, pages 319–335, Berlin, 2004. Springer Verlag.

19. H. Kastenberg and A. Rensink. Model Checking Dynamic States in GROOVE. In A. Valmari, editor, *Model Checking Software (SPIN), Vienna, Austria*, volume 3925 of *Lecture Notes in Computer Science*, pages 299–305, Berlin, 2006. Springer-Verlag.
20. P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 217–224. ACM, 2002.
21. T. Dimkov, W. Pieters, and P. Hartel. Laptop theft: a case study on the effectiveness of security mechanisms in open organizations. In *CCS '10: Computer and Communications Security*, pages 666–668, NY, USA, 2010. ACM.
22. B. Dragovic and J. Crowcroft. Containment: from context awareness to contextual effects awareness. In *Proceedings of 2nd Inernational Workshop on Software Aspects of Context*. CEUR Workshop Proceedings, 2005.
23. D.J. Scott. *Abstracting Application-Level Security Policy for Ubiquitous Computing*. PhD thesis, University of Cambridge, 2004.
24. S. Mathew, S. Upadhyaya, D. Ha, and H.Q. Ngo. Insider abuse comprehension through capability acquisition graphs. *Information Fusion, 2008 11th International Conference on*, pages 1–8, 30 2008-July 3 2008.
25. D. Gorla and R. Pugliese. Resource access and mobility control with dynamic privileges acquisition. In *Proc. of 30th International Colloquium on Automata, Languages and Programming (ICALP'03)*, pages 119–132. Springer Berlin / Heidelberg, 2003.
26. C. W. Probst, R. R. Hansen, and F. Nielson. Where can an insider attack? In *Workshop on Formal Aspects in Security and Trust (FAST 2006)*, pages 127–142. Springer, 2006.
27. Rocco De Nicola and Michele Loreti. A modal logic for mobile agents. *ACM Trans. Comput. Logic*, 5(1):79–128, 2004.
28. J.M. Wing. *Scenario Graphs Applied to Network Security*, chapter 9, pages 247–277. Morgan Kaufmann, 2007.
29. V. Mehta, C. Bartzis, H. Zhu, E. Clarke, and J. Wing. Ranking attack graphs. *LNCS*, 4219:127, 2006.
30. R.E. Sawilla and X. Ou. Identifying critical attack assets in dependency attack graphs. In *ESORICS '08: Proceedings of the 13th European Symposium on Research in Computer Security*, pages 18–34, Berlin, Heidelberg, 2008. Springer-Verlag.
31. O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. pages 273–284, 2002.
32. L. Wang, A. Singhal, and S. Jajodia. Measuring the overall security of network configurations using attack graphs. *LNCS*, 4602:98, 2007.

APPENDIX

Proof (of Theorem 1). The theorem follows from three properties, which we prove in turn:

1. There are no cycles between layers.
2. There are no cycles in the object layer.
3. There are no cycles in the digital layer.

1. There are no cycles between layers
   $\nexists\langle n_0...n_i...n_k\rangle : n_0 = n_k \wedge \mathcal{D}(n_0) \neq \mathcal{D}(n_i)$
   Lets assume that such a cycle exists:
   $\exists\langle n_0...n_i...n_k\rangle : n_0 = n_k \wedge \mathcal{D}(n_0) \neq \mathcal{D}(n_i)$
   Thus, there are at least two edges in the graph which connect nodes from different layers:
   $\exists(n_{j-1}, n_j), (n_l, n_{l+1}) \in Edge : \mathcal{D}(n_{j-1}) \neq \mathcal{D}(n_j) \wedge \mathcal{D}(n_l) \neq \mathcal{D}(n_{l+1}) \wedge \mathcal{D}(n_{j-1}) = \mathcal{D}(n_{l+1}) \wedge \mathcal{D}(n_j) = \mathcal{D}(n_l)$
   From the invariants 3, 4, 5 (tabulated in Table 1) follows that such a pair of edges does not exist.

Layer 1 (L_1)	Layer 2 (L_2)	Edge from L_1 to L_2	Edge from L_2 to L_1
L	O	+	- (invariant 3)
L	D	- (invariant 5)	- (invariant 5)
O	D	+	- (invariant 4)

Table 1. Invariants 3,4,5 forbid any cycles between layers.

2. There are no cycles in the object layer.
   $\nexists\langle n, ..., m\rangle : \mathcal{D}(n) = ... = \mathcal{D}(m) = O \wedge n = m$
   Lets assume such a cycle exists:
   $\exists\langle n, ...n_i..., m\rangle : \mathcal{D}(n) = ... \mathcal{D}(n_i) ... = \mathcal{D}(m) = O \wedge n = m.$
   From invariant 2,
   $\exists m \in Node : \mathcal{D}(m) = L \wedge \exists\langle m, ...n'_{i-1}, n_i\rangle$, follows
   $\exists(n'_{i-1}, n_i), (n_{i-1}, n_i)$. If $n'_{i-1} \neq n_{i-1}$ there is a contradiction with invariant 1. Otherwise $\mathcal{D}(n'_{i-1}) = O$, and the analysis is repeated for the path $\langle m, ...n'_{i-1}\rangle$. Because $\langle m, ...n'_{i-1}\rangle$ is finite, at one point the path reaches a spatial node, and $n'_{i-1} \neq n_{i-1}$. This again contradicts with invariant 1. Thus, such cycle does not exist.
3. There are no cycles in the digital layer.
   $\nexists\langle n, ..., m\rangle : \mathcal{D}(n) = ... = \mathcal{D}(m) = D \wedge n = m$
   This comes directly from invariant 6.

.

Proof (of Theorem 2). Suppose there is a net N_1 which satisfies the invariants $C(Map(N_1), \mathcal{D})$. Suppose exists a net N_2 which is a product of a net transformation on N_1. $\exists N_2 : N_1 \Rightarrow N_2$. We need to prove that $C(Map(N_2), \mathcal{D})$ also holds.

The relation $\Rightarrow$ is used in the net actions *neteval*, *netcopy* and *netmove*.

1. *neteval* does not cause any changes of the structure of the net. Thus any execution of *neteval* cannot invalidate an invariant.
2. *netmove* removes an edge (l_{t_1}, l) and generates a new one (l_{t_2}, l). We need to show that the $\xrightarrow{login(l,l_{t_2},l_o,s_{t_2})}$ action does not invalidate any invariant. Suppose the rule invalidates an invariant.
   (a) Let $\mathcal{D}(l) = O$. After $\xrightarrow{logout(l,l_{t_1},l_o,s_{t_1})}$, $indegree(l) = 0$. Latter, when $\xrightarrow{login(l,l_{t_2},l_o,s_{t_2})}$ is applied, $indegree(l) = 1$. Thus, invariant 1 is not invalidated.
   (b) Let $\mathcal{D}(l) = O$. After $\xrightarrow{login(l,l_{t_2},l_o,s_{t_2})}$ is applied, from $\succ_{ln}$, $\mathcal{D}(l_{t_2}) = L$ or $\mathcal{D}(l_{t_2}) = O$. The former case does not invalidate the second invariant by definition. Since $C(Map(N_1), \mathcal{D})$, $\exists m \in Node : \exists \langle m ... l_{t_2} \rangle \wedge \mathcal{D}(m) = S$, the latter case also does not invalidate the second invariant.
   (c) The invariants 3, 4, 5 are not invalidated by the definition of $\succ_{ln}$.
   (d) The last invariant is not invalidated because of the assumption in $\succ\succ$.
3. The effect of *netcopy* is an additional edge in the graph edge (l_t, l) generated by the relation $\xrightarrow{login(l,l_t,l_o,s_t)}$. The premise of *netcopy* enforces a restriction $\mathcal{D}(l_t) = D$. Additional restriction comes from the relation $\succ_{ln}$, which allows an edge to be generated only between a node from the object and digital layer $\mathcal{D}(l) = D \wedge \mathcal{D}(l_t) = O$ or between two nodes from the digital layer $\mathcal{D}(l) = D \wedge \mathcal{D}(l_t) = D$. The former does not invalidate any of the invariants, while the latter is restricted by the assumption on $\succ\succ$.