Portunes: Representing Attack Scenarios Spanning through the Physical, Digital and Social Domain

Trajce Dimkov, Wolter Pieters, and Pieter Hartel

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

Abstract. The security goals of an organization are realized through security policies, which concern physical security, digital security and security awareness. An insider is aware of these security policies, and might be able to thwart the security goals by combining physical, digital and social means. A systematic analysis of such attacks requires the whole environment where the insider operates to be formally represented. This paper presents Portunes, a framework which integrates all three security domains in a single environment. Portunes consists of a high-level abstraction model focusing on the relations between the three security domains and a lower abstraction level language able to represent the model and describe attacks which span the three security domains.

Using the Portunes framework, we are able to represent a whole new family of attacks where the insider is not assumed to use purely digital actions to achieve a malicious goal.

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. 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. If the environment is represented formally, it is possible to analyze potential insider attacks systematically.

The three security domains presented in the environment focus on different elements of security. Physical security restricts access to buildings, rooms and

A. Armando and G. Lowe (Eds.): ARSPA-WITS 2010, LNCS 6186, pp. 112–129, 2010.

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.

The majority of formal models for the insider threat assume the insider uses only digital means to achieve an attack. Therefore an essential part of the environment of interest is not captured. Indeed, a study performed by the National Threat Assessment Center in the US (NTAC) [2] shows that 87% of the attacks performed by insiders required no technical knowledge and 26% used 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 [3], in which the insider uses physical, digital and social means to compromise the asset cannot be presented nor analyzed. An example of a physically-enabled digital attack is the road apple attack [4], where an insider tricks an employee into plugging a malicious dongle into a server located in a physically restricted area. The road apple attack will be used as the main example in the paper.

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. Secondly, the domains have different properties making them hard to integrate. For example, mobility of digital data is not restricted by its locality as it is the case with objects in the physical domain. Likewise, physical objects cannot be reproduced as easily as digital data.

The contribution of this paper is the Portunes framework[1], a framework which integrates all three security domains in a single environment. Portunes consists of a model and a language. 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 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 attacks which span the three security domains.

The rest of the paper is structured as follows. Section 2 gives an overview of related work which contributed to the design of Portunes. Section 3 formalizes the Portunes model and Portunes language. We use the road apple attack as an example of the scenarios Portunes is designed to represent. The final section concludes and identifies future work.

2 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

[1] After Portunes, the Roman god of keys.

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. [5] 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 [6] focuses on 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 on the graph structure by giving meaning to the nodes and edges and defines invariants enforced directly into the semantics of the language.

KLAIM [7] is a process calculus for agent interaction and mobility, consisting of three layers: nodes, processes and actions. There are several KLAIM dialects, including μKlaim [8], OpenKlaim [9] and acKlaim [10]. The goal of the acKlaim language 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 Portunes language are similar to acKlaim and μKlaim. However, in the Portunes language mobility is represented by moving nodes rather than evaluating processes. Additionally, the Portunes language introduces delegation, whereby a node can delegate a task to another node.

3 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.

3.1 Requirements and Motivation

A model integrating multiple security domains needs to be expressive enough to present the details of an attack in each security domain. In a previous work [11], 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.

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

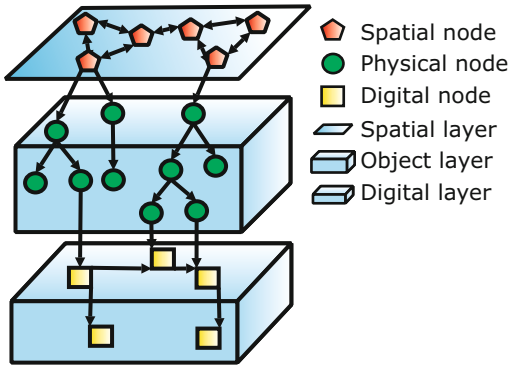

Fig. 1. Graphic presentation of Portunes

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.

3.2 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 graph. The model stratifies the nodes of the graph in three layers 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. 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.

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

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

Fig. 2. The ontology of Portunes

Definition 1. *Let $G = (Node, Edge)$ be a directed graph and $\mathcal{D} : Node \rightarrow Layer$ a function mapping a node to the $Layer = \{L, O, D\}$. 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* $\langle m, ..., n \rangle$ *denotes a finite path from* m *to* n*.*
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.*
   $\nexists \langle n, ..., m \rangle : \mathcal{D}(n) = ... = \mathcal{D}(m) = D \wedge n = m$

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:*
$\exists \langle n, ..., m \rangle : n = m \rightarrow \mathcal{D}(n) = ... = \mathcal{D}(m) = L$

Proof. The proof is presented in the appendix.

Example: Road apple attack. To show how Portunes can be used for representing insider threats across domains, we will use the example of the road

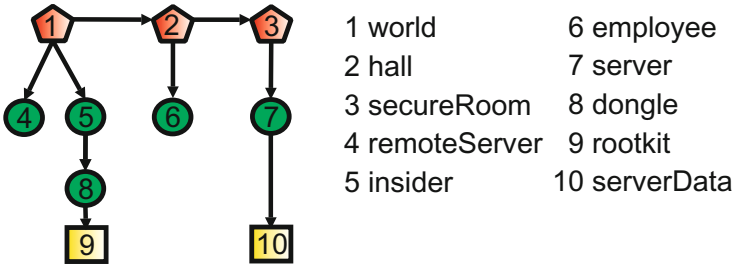

Fig. 3. Graph of the road apple attack environment

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

Fig. 4. The function $\mathcal{D}$ for the road apple attack environment

apple attack [4]. In this attack, an insider uses the trust of an employee (social domain) to steal sensitive data (digital domain) from a a server in a restricted area (physical 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).

An abstraction of the environment is represented as a Portunes model in Figure 3 and 4. The nodes *hall*, *secureRoom* and *world* are spatial nodes, *serverData* and *rootkit* are digital nodes. All other nodes are object nodes. In Section 3.4 we will revisit the example and show how the road apple attack takes place.

3.3 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, which is closer to the enforcement mechanisms. The language consists of nodes, processes and actions, where a node in the Portunes model represents a node in the Portunes language. The main goal of the language is to model the interaction between the nodes in the Portunes model.

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 his initial privileges. The Portunes language lets a delegator node delegate a task to a delegatee node. During the execution of the task, the delegatee uses the privileges of the delegator. To delegate a task, the delegatee needs to trust the delegator. For example, an insider can delegate a task to a colleague. The colleague will execute the task only if he trusts the insider.

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.

Syntax. 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 5. A single *node* $l ::_s^\delta P$ consists of a name $l \in \mathcal{L}$, where $\mathcal{L}$ is a finite set of names, a set of node names $s \in \mathcal{P}(\mathcal{L})$, 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

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

Fig. 5. Syntax of the Portunes language

connects l to a node name l' in the set 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.

A *process* P is a composition of actions. Namely, *nil* stands for a process that cannot execute any action and $a^l.P$ for the process that executes action a using privileges from node $l \in \mathcal{L}$ and then behaves as P. 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.

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 the node.

Example: For a node representing a room, $room ::_s^\delta nil$, the access control policy δ defines the conditions under which other entities can enter or leave the room. The set s contains the names of all nodes that are located in the room or connected to the room. Let a supervisor and a person be in a hall $hall ::_{\{person,\ supervisor\}}^\delta nil$ which is neighboring the room. An example of a supervisor delegating a task to a person is: $supervisor ::_s^\delta eval(P)@person^{supervisor}$ where P is a process denoting the task, *person* is the target node and the label *supervisor* is the origin node. A person entering the room as part of the task delegated from *supervisor* is presented through $person ::_s^\delta login(room)^{supervisor}.P'$, while a person leaving the room $person ::_s^\delta logout(room)^{supervisor}.P''$.

Depending on the privileges of the origin node which depend on its identity, location and credentials, a node can grant a set of capabilities $C = \{ln, lt, e\}$, where ln is a capability to execute the action *login*, lt to execute the action *logout* and e to execute the action *eval*. The access control policy δ is a function $\delta : (\mathcal{L} \cup \{\bot\}) \times (\mathcal{L} \cup \{\bot\}) \times \mathcal{P}(\mathcal{L}) \rightarrow \mathcal{P}(C)$. 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 least restrictive policy that can be used is: $(\bot, \bot, \emptyset) \mapsto \{ln, lt, e\}$.

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

$$l_t \succ_{ln} l = \begin{cases} true & iff\ \ (\mathcal{D}(l_t) = L \wedge \mathcal{D}(l) = O) \vee (\mathcal{D}(l_t) = O \wedge \mathcal{D}(l) = D) \\ l_t \succ\succ_{ln} l & iff\ \ \mathcal{D}(l_t) = \mathcal{D}(l) \\ false & otherwise \end{cases}$$
$$\text{where } l \succ_e l_t =$$
$$\underbrace{(\mathcal{D}(l) \neq L \wedge \mathcal{D}(l_t) \neq L)}_{(4)} \wedge \underbrace{\neg(\mathcal{D}(l) = D \wedge \mathcal{D}(l_t) = O)}_{(5)} \wedge \underbrace{(l_t \in children(l)}_{(6)}$$
$$\vee \underbrace{(\exists l_p ::^{\delta_p}_{s_p} R \in N : l \in s_p \wedge l_t \in s_p)}_{(7)} \vee \underbrace{\mathcal{D}(l_t) = D)}_{(8)}$$

Fig. 6. Auxiliary function *grant* and $\succ$ relations

Auxiliary functions. Having defined the behavior of nodes using three primitive actions, 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 three reasons. The origin node might not have sufficient privileges, execution of the action a invalidates the invariants in Definition 1 from the Portunes model, or the target node might not be in proximity of the node l. This section defines auxiliary functions for an implicitly given net N, which take care these restrictions. The auxiliary functions are defined in Figure 6 and are used to simplify the operational semantics of the language.

The *grant* function checks if an origin node has sufficient privileges to execute an action to a target node. The first parameter defines the name of the origin node, the second parameter defines the policies on the target node and the third parameter is a label of an action. Intuitively, 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 node l. The goal of this relation is to enforce the invariants 3-6 in Definition 1. From the relation, an object node can always interact with spatial nodes and a digital node can always interact with object nodes. The relation $l_t \succ\succ_{ln} l$ provides ordering between nodes from the same layer. The relation is defined by the user because the ordering depends on the elements we want to model in the environment. For example, an operating system usually can contain a file, but not vice versa. Yet, in scenarios where the systems are virtualized, it is possible and desirable to model a file containing an operating system. The only assumption on $l_t \succ\succ_{ln} l$ is that it does not invalidate invariant 7 in Definition 1, or put differently, the relation does not allow generation of cycles between nodes in the digital layer.

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 6 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 which helps determine if a process is a simple process.

Definition 1. *Let* $origin(P) \rightarrow \mathcal{P}(\mathcal{L})$ *be a function which returns all the action labels of a process* P*. A process* P*, which is either nil or contains actions only from one origin node is a simple process.* $origin(P) \subseteq \{l_0\}$

Operational semantics. Similar to Bettini et al. [9], 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 are the target node contains. The net semantics given in terms of a transition relation $\Rightarrow$ describes possible net evolutions and relies on the labeled transition $\xrightarrow{a}$ from the process semantics.

The process semantics of the language is defined in Figure 7. 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 in 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

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

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

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

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

Fig. 7. Process semantics

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

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

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

Fig. 8. Net semantics

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.

The net semantics in Figure 8 use 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 proximity, 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 entering a new node without leaving the previous **[netcopy]**. The standard rules for structural congruence apply and are presented in Figure 9.

Theorem 1. *Nodes from the object and spatial layer cannot move to remote locations.*

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

Theorem 2. *Nodes from the object and spatial layer can influence only child and sibling nodes.*

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

Fig. 9. Structural congruence of processes and nets

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

Theorem 3. *Let G be a Portunes graph and N be a network of nodes in Portunes language. Let $Map(N) \rightarrow G$ map a Portunes program in a Portunes model, such that $C(Map(N), \mathcal{D})$ holds.*

The transitions generated from the semantics of Portunes language do not invalidate $C(Map(N), \mathcal{D})$.

Proof. The proof is presented in the appendix.

3.4 Using the Portunes Framework to Calculate Attack Scenarios

Having defined Portunes in the previous sections, this section shows how the framework can aid in calculating attack scenarios. The Portunes model helps represent the environment graphically and puts constraints on structure. The user needs to define: (1) a net composition that corresponds to the graph with variables instead of processes, (2) the function $\mathcal{D}$, which stratifies the graph, and (3) the relation $\succ\succ_{ln}$ which tells which node can be contained in which other node.

l / l_t	1	2	3	4	5	6	7	8	9	10
1. world										
2. hall										
3. secureRoom										
4. remoteServer								1		
5. insider				1			1	1		
6. employee				1			1	1		
7. server								1		
8. dongle										
9. rootkit										
10. serverData									1	

Fig. 10. Definition of the auxiliary relation $\succ\succ_{ln}$ for the road apple attack environment

The previous steps provide a representation of the environment of interest. It is now possible to present attack scenarios through process definitions. The last step (4) is to find concrete process expressions (i.e.instantiations of the variables in item (1) that invalidate a goal. An attack scenario can be generated by hand or automatically, by using model checking techniques. Here we use the road apple attack as an example of an attack scenario.

Example: Road apple attack - continued. In section 3.2 we introduced the Portunes model of the environment where the road apple takes place. We defined the relation between the elements through a graph and their properties through the function $\mathcal{D}$. Now, we additionally define the $\succ\!\!\succ_{ln}$ relation and the security policies on each of the nodes.

The relation $\succ\!\!\succ_{ln}$ is defined in Figure 10 through a boolean table. For example, cell (4,8) is the result of $remoteServer \succ\!\!\succ_{ln} dongle$ and indicates that the remote server can contain the dongle.

Figure 11 presents the environment as a net composition. This representation does not provide visual information about the relation between elements, as in the Portunes model. However, the representation contains detailed information about the security policies in place, making it suitable for analysis.

Having defined the environment, now it is possible to reason about possible attack scenarios. An attack scenario is defined through generating processes in the nodes. Figure 12 shows the dynamics 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

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

Fig. 11. The road apple attack environment in the Portunes language

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

Fig. 12. The road apple attack in the Portunes language

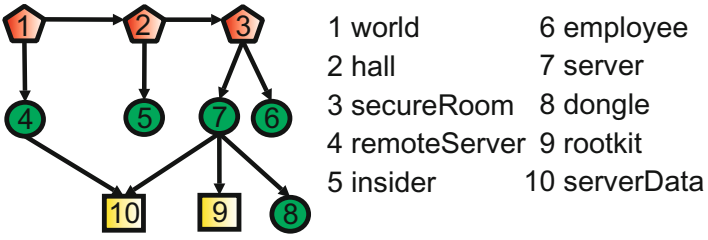

Fig. 13. Portunes model of the road apple attack environment after the execution of the attack

origin node are omitted from the process definitions. 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 12, the data will reside in the remote server, presented through an edge $(remoteServer, data)$ in the Portunes model in Figure 13.

The process definitions follow the semantics of the language. Thus, no attack defined through processes will violate a security policy. This makes the framework suitable for presenting scenarios where the insider does not violate a policy, but achieves his goal by combining physical access, social engineering and digital actions.

The road apple attack is just one attack scenario. An insider may gain possession of the data by using alternative routes. For example, the employee might be tricked into letting the insider in the secure room, as shown through the process definitions in Figure 14. A proper reasoning about the data exposure requires all attack scenarios to be available to the security professional. The Portunes framework aids in the reasoning of data exposure, by helping answer questions such as:

1. In which locations can an object A end up? For example, show all locations where the server data can reside.
2. Who can reach location A? For example, show all elements who can reach the secure room.
3. What are the scenarios that violate a specific goal? For example, show all attack scenarios where the server data ends up in a remote server.

To answer these questions, we implemented a proof of concept implementation of the framework and used model checking to generate all possible attack scenarios by automatically generating the processes P_1 - P_4. However, model checking requires heuristics to improve the scalability and we are currently exploring other

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

Fig. 14. Alternative attack scenario

techniques for the generation of attack scenarios. We will discuss the algorithms in more detail in future work.

4 Conclusion and Future Work

This paper presents Portunes, a framework consisting of a high-level model and a language inspired by the KLAIM family of languages. Portunes is capable of representing attacks spanning the digital, physical and social domain. To capture the three domains efficiently, Portunes is able to represent 1) physical properties of elements, 2) mobility of objects and data, 3) identity, credentials and location based access control and 4) trust and delegation between people. 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.

As a future work, we plan to generate attack scenarios automatically from environments presented through the Portunes framework. We are looking at existing model checking techniques and heuristics to generate all possible action traces for each of the processes. Additionally, we are interested in mechanisms to isolate actions which contribute to an attack and automatically generate attack trees.

References

1. INFOSEC Research Council. Hard problem list (November 2005), `http://www.cyber.st.dhs.gov/docs/IRC_Hard_Problem_List.pdf`
2. Randazzo, M.R., Keeney, M., Kowalski, E., Cappelli, D., Moore, A.: Insider threat study: Illicit cyber activity in the banking and finance sector. U.S. Secret Service and CERT Coordination Center Software Engineering Institute (2004)
3. DePoy, J., Phelan, J., Sholander, P., Smith, B.J., Varnado, G.B., Wyss, G.D., Darby, J., Walter, A.: Critical infrastructure systems of systems assessment methodology. Technical Report SAND2006-6399, Sandia National Laboratories (October 2007)
4. Stasiukonis, S.: Social engineering the usb way (2006), `http://www.darkreading.com/document.asp?doc_id=95556`
5. Dragovic, B., Crowcroft, J.: Containment: from context awareness to contextual effects awareness. In: Proceedings of 2nd Inernational Workshop on Software Aspects of Context. CEUR Workshop Proceedings (2005)
6. Scott, D.J.: Abstracting Application-Level Security Policy for Ubiquitous Computing. PhD thesis, University of Cambridge, Cambridge (2004)

7. De Nicola, R., Ferrari, G.L., Pugliese, R.: KLAIM: A kernel language for agents interaction and mobility. IEEE Transactions on software engineering 24(5), 315–330 (1998)
8. Gorla, D., Pugliese, R.: Resource access and mobility control with dynamic privileges acquisition. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 119–132. Springer, Heidelberg (2003)
9. Bettini, L., Loreti, M., Pugliese, R.: An infrastructure language for open nets. In: SAC 2002: Proceedings of the 2002 ACM Symposium on Applied Computing, pp. 373–377. ACM, New York (2002)
10. Probst, C.W., Hansen, R.R., Nielson, F.: Where can an insider attack? In: Dimitrakos, T., Martinelli, F., Ryan, P.Y.A., Schneider, S. (eds.) FAST 2006. LNCS, vol. 4691, pp. 127–142. Springer, Heidelberg (2007)
11. Dimkov, T., Tang, Q., Hartel, P.H.: On the inability of existing security models to cope with data mobility in dynamic organizations. In: Proceedings of the Workshop on Modeling Security. CEUR Workshop Proceedings (2008)
12. AlZarouni, M.: The reality of risks from consented use of usb devices. In: Valli, C., Woodward, A. (eds.) Proceedings of the 4th Australian Information Security Conference, pp. 5–15 (2006)

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.

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

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)

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 3). 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 $\succ\succ$.