Model Checking of Location and Mobility Related Security Policy Specifications in Ambient Calculus

Devrim Unal[1], Ozan Akar[2], and M. Ufuk Caglayan[2]

[1] TUBITAK National Research Inst. of Electronics and Cryptology
[2] Bogazici University

Abstract. Verification of security for mobile networks requires specification and verification of security policies in multiple-domain environments. Mobile users present challenges for specification and verification of security policies in such environments. Formal methods are expected to ensure that the construction of a system adheres to its specification. Formal methods for specification and verification of security policies ensure that the security policy is consistent and satisfied by the network elements in a given network configuration. We present a method and a model checking tool for formal specification and verification of location and mobility related security policies for mobile networks. The formal languages used for specification are Predicate Logic and Ambient Calculus. The presented tool is capable of spatial model checking of Ambient Calculus specifications for security policy rules and uses the NuSMV model checker for temporal model checking.

Keywords: model checking, ambient calculus, security policy.

1 Introduction

Resource sharing and provision of services in networks with multiple administrative domains is an ever-increasing need. Roaming is another concept that comes into consideration when dealing with multi-domain resource sharing applications where users are allowed to use network connectivity of multiple domains. Roaming means that users are able to connect to and use networks of multiple administrative domains. Security management in such an environment requires specification of inter-domain security policies and cross-domain administration of security mechanisms. Authorization mechanisms determine the access rights for a user based on the security policy. The access control mechanisms then control the user access to the resource based upon these determined access rights. The user actions should be verified against home and visited domain policies as they access resources on visited domains. Formal verification can be used to ensure that visiting users are not bypassing security mechanisms and violating security policy by making use of the internal trust relationships.

In this paper, we propose a method and a model checking tool for formal specification and verification of multi-domain security policies with location and

I. Kotenko and V. Skormin (Eds.): MMM-ACNS 2010, LNCS 6258, pp. 155–168, 2010.

mobility constraints. Appropriate to the nature of multi-domain mobile networks, the proposed method focuses on the location and mobility aspects of security policies. The formalism of the method is based on predicate logic, ambient calculus and its ambient logic. An ambient calculus model checker capable of spatial and temporal model checking has been built for the implementation of proposed method. The model checker presents novel computational methods for decreasing the time and space complexity of spatial model checking. The model checking approach complements our previous work on use of theorem proving for security policies [19].

2 Formal Languages and Methods for Specification and Verification of Policies

Logic-based security policy models provide a general framework for security policy specification. Becker et al.'s SECPAL [1] is a formal security policy language for Grid environments. The Flexible Authorization Framework (FAF) is a logic programming based method for definition, derivation and conflict resolution of authorization policies [14,13]. Another study based on logic that supports explicit denials, hierarchies, policy derivation and conflict resolution is [2]. Ponder [9] is a general purpose formal security policy language. Woo and Lam [20] define a paraconsistent formal language for authorizations based on logical constructs. In [8] deontic logic is used for modeling the concepts of permission, obligation and prohibition with organizational constructs. A security policy language based on the set-and-function formalism is presented in [16].

Model checking and theorem proving have been applied for verification of security policies. Acpeg [21] is a tool for evaluating and generating access control policies based on first-order logic. We have previously applied theorem proving to verification of security policies. In [19] we use Coq for checking that an authorisation security policy is conflict-free, initially and as authorisation rules are added and removed, while [10] uses first order linear temporal logic embedded within Isabelle to formalise and verify RBAC authorisations constraints. A more recent study, [18] uses nontemporal and history-based authorization constraints in the Object Constraint Language (OCL) and first-order linear temporal logic (LTL) to verify role-based access control policies with the help of a theorem prover.

Ambient Calculus [4] has been used for modeling and reasoning about security in mobile systems. Reasoning about spatial configurations for application level security policies in ubiquitous environments is one of the issues investigated in Scott's PhD thesis [17]. In this study a simplified version of ambient calculus and ambient logic is used in policy rules of a security policy. BACI_{R} [7] is a boxed ambient calculus with RBAC mechanisms used to define access control policies for ambients. Similarly, in our approach, the Ambient Calculus and Ambient Logics [3] are utilized. In contrast to BACI_{R} which places policies inside Ambient Calculus formulas, we use Ambient Calculus for specification of processes and Ambient Logics for specification of policies and complement them with Predicate

Logic based relational model. In contrast to Scott's approach, network level policies rather than application level policies will be covered and locations will denote placement in domains and hosts. For model checking of ambient calculus specifications, our approach is similar to the work of Mardare et al [15]. We use a modified version of Mardare algorithm and present an algorithm based on use of capability trees that reduces complexity of state space generation and matching of ambient logic formulas to states. In the works of Charatonik et al. [5,6] exhaustive search is offered for searching possible decompositions of processes and searching sub locations when checking spatial modalities. We offer heuristics for searching possible decompositions of processes and searching sub locations to reduce the search space.

3 A Formal Model for Security Policies for Multi-domain Mobile Networks in Ambient Calculus

3.1 Formal Model for Security Policy

Access Control Model: The access control model is specified using Predicate Calculus and First Order Set Theory. The access control model is based on the RBAC [11] model. The Hierarchical RBAC model extends the Core RBAC model with role hierarchies. We use the hierarchical RBAC model that supports role hierarchies. For introduction of location and mobility constraints into the security policy, we extend the Hierarchical RBAC model by adding Domains, Hosts, Object Types, Conditions and Location Constraints. The concept of sessions are not utilized in our model.

- Constants:
  d, n, m, o, t, v: Number of domains, hosts, users, roles, objects and object types, respectively.
- Sets:
  - $D = \{D_1, D_2, ...D_d\}$: Domains , $H = \{H_1, H_2, ...H_n\}$: Hosts
  - $U = \{U_1, U_2, ...U_m\}$: Users , $R = \{R_1, R_2, ...R_o\}$: Roles
  - $O = \{O_1, O_2, ..., O_t\}$: Objects , $OT = \{OT_1, OT_2, ...OT_v\}$: Object Types
- Relations:
  $HOD : H \times D$: Maps hosts to domains. $HOD(H_i, D_a)$ denotes that H_i is enrolled to Domain D_a.
  $UOD : U \times D$: Maps users to domains. $UOD(U_j, D_a)$ denotes that U_j is enrolled to Domain D_a.
  $OOT : O \rightarrow OT$:Function that specifies the type of an object. $OOT(O_k)$ gives the type of object O_k.
  $UA : U \times R$: Relation for assignment of users to roles.
  $PA : R \times AO \times SA$: Relation for associating roles with permissions.

Authorization Terms and Security Policy
An Authorization Term is of the form $at = (as,\ ao,\ sa,\ fo,\ co)$ where $as \in$ AS, $ao \in$ AO, $sa \in S \times A$,fo: a formula, where formula is an ambient logic formula, co: a condition, where condition is a predicate logic formula. Security policy is a set of authorization terms.

- Sets:
  AS: $AS = U \cup R$. The set of Authorization Subjects. Authorization Subjects are active entities that may conduct an Action on an Authorization Object.
  AO: $AO = O \cup OT \cup H \cup D$. The set of Authorization Objects. The authorization object is the entity upon which an action is conducted.
  A : Set of actions conductable by subjects on objects. We take A to be fixed in this study:
  A =*{Enroll, Login, Logout, Execute, Read, Write, Send, Receive, Delete, Create}*
  Signs: $S = \{+, -\}$ Represents permission or denial.
  Signed Actions: $S \times A$: Represents permission or denial of an action (+,read) denotes that read action is permitted.
- Predicates:
  - EnrolledDomainHost (*host*, *domain*): *host* is a registered member of the Domain *domain*
  - EnrolledDomainUser (*user*, *domain*): *user* is a registered member of *domain*
  - ActiveDomainUser (*user*, *domain*): *user* has logged into a *domain*
  - RoleAllowed (*user*, *role*): *user* has assumed the Role of *role*
  - ActionAllowed (*as*, *action*): Authorization Subject *as* is allowed to execute action *action*
- Conditions: First-order sentences built on the Predicates defined above.
- Spatial Formula: Ambient Logic formula that includes names of domains, authorization subjects and authorization objects. The formula will be described in the following sections.

3.2 Formal Specification of Mobile Processes

Ambient calculus, proposed by Cardelli and Gordon, is a process calculus which is able to theorize about concurrent systems that include mobility and locations [4]. The proposed methodology uses ambient calculus for specifying multi-domain mobile network configurations. Fragment of ambient calculus used in this paper is shown at Table 1. The semantics of ambient calculus is based on structural congruence relation.

The formal model for mobility is a finite fragment of the ambient calculus with public names as used in [6]. In the formal specification, domains, hosts, users and objects are modeled as Ambients. The actions are modeled as Ambient Calculus capabilities. A process specification shows a trace of a process in a certain mobile network scenario. Each scenario may be modeled as a set of process specifications. These specifications will then be checked against a security policy for compliance. The process specification involves capabilities, objects and ambients. Resources may be *input* and *output* by the ambients. The ambients may be World, Domains, Hosts and Users. Below some examples of object, host and user mobility specification of mobility as ambient calculus processes are listed. Some known notation conventions are utilized: for example $n[]$ means $n[0]$. The symbol $\rightarrow$ represents the reduction relation and $\rightarrow^*$ represents a series of reductions.

Table 1. Mobility and communication primitives of Ambient Calculus

P, Q	::=	processes	M	::=	capabilities
	0	inactivity		x	variable
	$P\|Q$	composition		n	name
	$M[P]$	ambient		$in \quad M$	can enter M
	$M.P$	capability		$out \quad M$	can exit M
	$(x).P$	input		$open \quad M$	can open M
	$\langle M \rangle$	asynchronous output		ϵ	null
				$M.M$	path

– *File1* is copied to *Portable1*:

$$\begin{gathered} \mathit{World}[\mathit{DomainA}[\mathit{Server1}[\text{folder }[\text{out folder. out } \mathit{Server1}.\text{ in } \mathit{Portable1}.\text{ in} \\ \text{folder. } \mathit{File1}[] \mid \mathit{File1}\ []]] \mid \mathit{Portable1}[\text{folder}[]]]] \rightarrow^* \\ \mathit{World}[\mathit{DomainA}[\mathit{Server1}[\text{folder }[\mathit{File1}[]]] \mid \mathit{Portable1}[\text{folder}[\mathit{File1}[]]]]] \end{gathered}$$

– A message M is sent from *User1* to *User3*:

$$\begin{gathered} \mathit{World}\ [\mathit{DomainA}\ [\mathit{Server1}\ [\mathit{User1}\ [\text{message}[M \mid \text{out } \mathit{User1}.\text{ out } \mathit{Server1}. \\ \text{out } \mathit{DomainA}.\text{ in } \mathit{DomainB}, \text{ in } \mathit{Client2}.\text{ in } \mathit{User3}.0]]]] \mid \mathit{DomainB}\ [\mathit{Client2} \\ [\mathit{User3}\ [\text{open message}.(m).0]]]] \rightarrow^* \\ \mathit{World}\ [\mathit{DomainA}[\mathit{Server1}[\mathit{User1}[]]] \mid \mathit{DomainB}[\mathit{Client2}[\mathit{User3}[M]]]] \end{gathered}$$

We also provide the mapping of actions in the security policy model to Ambient Calculus specifications. These are provided as a template and based on inference of specific subject and object names from the high-level specifications of security policy, the model checking tool is presented with suitable Ambient Calculus specifications.

$$\mathit{Enroll} =_{\text{def}} \mathit{newz.indomain.z}[]|\mathit{domain}[], \mathit{wherez} \in U \cup H, \mathit{domain} \in D \quad (1)$$

$$\mathit{Login} =_{\text{def}} \mathit{domain}[z[\mathit{inhost}]|\mathit{host}[]], \mathit{wherez} \in U, \mathit{host} \in H, \mathit{domain} \in D \quad (2)$$

$$\mathit{Logout} =_{\text{def}} \mathit{domain}[\mathit{host}[z[\mathit{outhost}]]], \mathit{wherez} \in U, \mathit{host} \in H, \mathit{domain} \in D \quad (3)$$

3.3 Formalization of Location and Mobility Related Actions in Authorization Term

The spatial formula in the Authorization Term is specified using the Ambient Logic[3]. Fragment of ambient logic used in this paper is shown in Table 2. Ambient logic has temporal and spatial modalities in addition to propositional logic elements. Semantics of the connectives of the ambient logic are given through satisfaction relations defined in [3]. The definition of satisfaction is based heavily on the structural congruence relation. The satisfaction relation is denoted by $\models$ symbol. To express that process P satisfies the formula $\mathscr{A}$, $P \models \mathscr{A}$ is used. The

Table 2. Syntax of ambient logic

η		a name n		
$\mathscr{A}, \mathscr{B}, \mathscr{C}$ ::=	T	true	$\mathscr{A} \mid \mathscr{B}$	composition
	$\neg\mathscr{A}$	negation	$n[\mathscr{A}]$	location
	$\mathscr{A} \vee \mathscr{B}$	disjunction	$\Diamond\mathscr{A}$	sometime modality
	0	void	$\diamond\mathscr{A}$	somewhere modality

symbol Π denotes the set of processes, Φ denotes the set of formulas, ϑ denotes the set of variables, and Λ denotes the set of names.

The possibility of conflicts arising of conflicting actions are resolved using the theorem prover as presented in our previous work in [19] before presenting rules to the model checker. Using the formalization methodology for authorization terms and spatial formula described above, some example security policy definitions with location constraints, which can be specified with our formal authorization terms are presented below.

1. All allowed users can read files in folder *Project_ Folder*, if they are in a location that contains this folder: (as = *, ao = *Project_ Folder*, sa = + *read*, co = *ActionAllowed* (*as*, *sa*), fo = $\diamond$(*as*[] | *ao*[]))
2. All allowed users can send *E-mail* between the *UniversityA* and *UniversityB* domains: (as = *, ao = *E-mail*, sa = + *send*, co = *ActionAllowed* (*as*, *sa*), fo = *UniversityA* [$\diamond$*as*[]] | *UniversityB* [$\diamond$*ao*[]] \/ *UniversityB* [$\diamond$*as*[]] | *UniversityA*[$\diamond$*ao*[]])

To check location constraints in security policy, the input to the model checker tool is an Ambient Calculus specification and a set of Ambient Logic formulas. An example scenario specified in Ambient Calculus and a security policy rule specified in Ambient Logic is presented below. In this example there are two domains, *Domain1* and *Domain2*, where *User2* is mobile and tries to read data from *File1* by logging into *Host1*. Spatial formula in the policy rule states that *Host2* can not contain *Data1* and *Data2* at the same time. This is a rule that means *Domain2* data should not be copied to *Domain1*.

- Ambient Calculus Specification: *Domain1* [*User1*[] | *Host1* [*File1* [*Data1* [in *User2*.0 | out *User2*.0]]]] | *Domain2* [*Host2* [*User2* [out *Host2*.0 | out *Domain2*.0 | in *Domain1*.in *Host1*.0 | out *Host1*.out *Domain1*.0 | in *Domain2*.in *Host2*.0 in *File1*.0 | in *File2*.0 | out *File1*.0 | out *File2*.0] | *File2*[*Data2*[]]]]
- Ambient Logic Specification: □ { ¬ ◇ { ◇ *Host2*[◇ {*Data1*[T] | *Data2*[T]}] | T}

4 Model Checking of Security Policy Specifications in Ambient Calculus Model Checker

The general structure of the Ambient Calculus model checker is given in Figure 1. To benefit from existing methodologies we divide our problem into two sub

problems as temporal model checking and spatial model checking. The temporal model checker is used for carrying out satisfaction process for the Sometime and Everytime connectives of ambient logic. The proposed model checking method generates all possible future states and build a state transition system based on the Ambient Calculus process specification. After evaluation of Ambient Logic formula in each state, this state transition system is processed into a Kripke Structure (Definition 4) which is then given to temporal model checker. NuSMV [12] is used as a temporal model checker. Outline of the proposed algorithm for the model checking problem is below.

1. Define atomic propositions with respect to spatial properties of ambient logic formula and register the (atomic proposition-spatial modality) couples.
2. Reduce ambient logic formula to temporal logic formula (CTL) by replacing spatial modalities with atomic propositions.
3. Generate state transition system of the ambient calculus specification with respect to reduction relations. This involves generation of initial state from given ambient calculus specification, generation of new states by applying available capabilities with respect to ambient calculus reduction relations and addition of new states to state transition system with transition relation.
4. Generate Kripke Structure from state transition system. This step involves the assignment of the values of the atomic propositions for each state of state transition system (labeling) by applying model checking for spatial modalities on ambient topology of the related state and the addition of a new state with its label (values of atomic propositions) to the Kripke Structure.
5. Generate NuSMV code from Kripke Structure and CTL Formula.

4.1 Ambient Topology and Spatial Formula Graphs

In [15], state information is represented with sets. In [6], calculus and logic information is represented as strings and algorithms are based on string operations. In the method proposed, ambient calculus specifications and logic formulas are represented as graphs. State information associated with a process specified in ambient calculus consists of static and dynamic properties. Static properties of state are the ambients and their hierarchical organization, i.e. the "ambient topology". The dynamic properties of the state are the capabilities and their dependencies on each other. Static and dynamic properties of an ambient calculus specification are kept in separate data structures.

Definition 1. *Ambient Topology,* $G_{AT} = (N_{AT}, A_{AT})$*, is an acyclic digraph where elements of set of nodes* $v \in N_{AT}$ *denotes ambients within the ambient calculus specification (elements of* Λ*) and arcs* $a \in A_{AT}$*,* $a = \{xy \mid x, y \in N_{AT}\}$ *denotes parent-child relation among ambients. The indegree of nodes* $deg^-(v) = 1$ *for any node (vertex)* v *whereas the outdegree of nodes* $deg^+(v) \in \mathbb{N}$*.*

The following defines capability trees which is a novel data structure used in our algorithm.

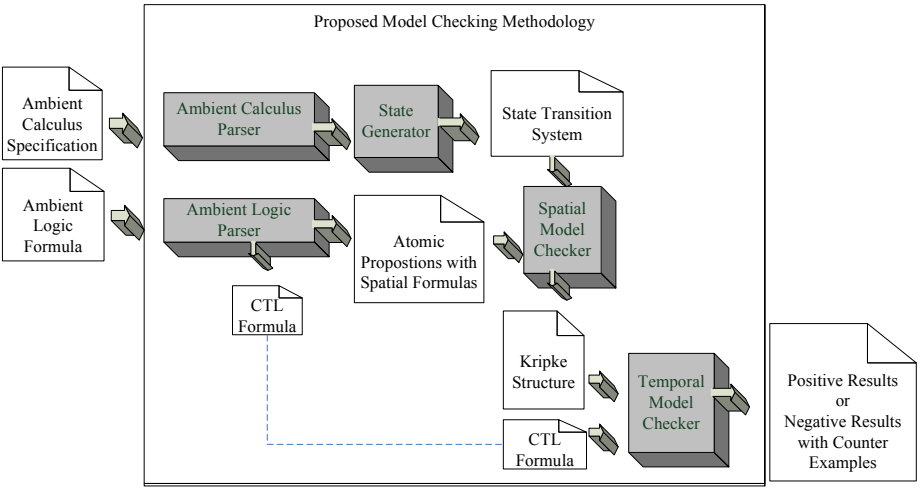

Fig. 1. Block diagram of the Ambient Calculus Model Checker

Definition 2. *Capability Tree,* $G_{CT} = (N_{CT}, A_{CT})$*, is an acyclic digraph where set of nodes* $v \in N_{CT}$ *denotes capabilities and arcs* $a \in A_{CT}$*,* $a = \{xy \mid x,\ y\ \in N_{CT}\}$ *denotes priority relation among capabilities. Nodes contain the information about which ambient the capability is attached and which ambient the capability effects.* $deg^-(v) = 1$ *for any node* v*, whereas* $deg^+(v) \in \mathbb{N}$*.*

Graphs representing formulas are more complex than the others. They are acyclic digraphs where nodes denote connectives and locations whereas arcs denote the operator-operand relation. There are multiple types of nodes and arcs in formula graphs because of the different structure of the ambient logic connectives.

Definition 3. *An ambient logic formula,* $G_F = (N_F, A_F)$*, is an acyclic digraph where*

- The set of nodes: $N_F = (N_L \cup N_{Binary} \cup N_{Unary} \cup N_{PC})$. N_L is the set of nodes representing ambients. Elements of N_L are labeled with elements of Λ. N_{Unary} is the set of nodes representing unary connectives ($\neg$, $\diamond$, $\Diamond$) at formulas. N_{Binary} is the set of nodes representing binary connectives, ($\vee$) at formulas. N_{PC} is the set of nodes representing parallel compositions at formulas.
- The set of arcs: $A_F = (A_{PC} \cup A_{Binary} \cup A_{Unary})$, where elements of A_{PC} represents parallel compositions, A_{Binary} represents binary connectives and A_{Unary} represents unary connectives of ambient logic formulas.
- $a_{pc} \in A_{PC} = x, y | x \in N_{PC},\ y \in (N_L \cup N_{Binary} \cup N_{Unary})$, $a_u \in A_{Unary} = (x, y \mid x \in (N_L \cup N_{Unary}), y \in N_{PC})$, $a_b \in A_{Binary} = (x, y | x \in N_{Binary},\ y \in N_{PC})$

- for $v \in N_F$, $deg^-(v) = 1$, for $v \in N_{PC}$, $deg^+(v) \in \mathbb{N}$, for $v \in N_{Unary}$, and $v \in N_L$, $deg^+(v) = 1$, for $v \in N_{Binary}$, $deg^+(v) = 2$.
- Elements of N_{PC} can have a special attribute to represent the **T** construct of the logic. If **T** attribute of a N_{PC} node is set to true this means the parallel composition of process that the N_{PC} node stands for, includes the constant **T**.

4.2 State Transition System Generation

In the proposed model checking methodology the state transition system is generated from the initial model specification by executing capabilities in the ambient calculus specification. Since replication is excluded from specifications, the state transition system can be represented by an acyclic digraph where nodes represent states and edges represent the execution of a capability. For selection of the next capability to execute, some condition checks are carried out. These conditions are the location of the object ambient and the availability of the subject ambient. A capability can not be executed if the location of the object ambient for the capability is not the current location, if it is prefixed by another capability path, or the parent ambient of the subject ambient is prefixed by a capability path. In the proposed method these conditions are checked each time a capability is to be executed.

In this work a new data structure is offered to represent temporal behaviors. The use of this data structure named "capability trees" eliminates the need to check the availability of a subject ambient. Capability paths are organized as an acyclic digraph that represent the interdependencies of capabilities. Capability trees are built at parsing stage so no pre-processing is needed. The selection of the next capability to execute starts from the root of this graph. This method guarantees that the capabilities of the parent processes are executed before the capabilities of child processes.

4.3 Checking Spatial Modalities

The basic element for building an ambient calculus model checker for ambient logic is to express and implement the satisfaction relation. In the proposed method, all the generated states generated must be checked against the spatial formulas. Ambient logic formulas are decomposed into a CTL formula and a set of spatial formulas by formula reduction. The ambient topology and the spatial formula graphs are inputs to the spatial model checker. The spatial model checking takes place before generation of Kripke Structures.

Matching of an ambient topology and a spatial formula is a recursive procedure in which ambient topology nodes are assigned to formula nodes. Matching process starts with assigning the ambient topology's root to the root of the spatial formula graph. Spatial formula nodes can forward the assigned ambient topology node to its children partially or completely in a recursive manner. Match process is successful when all nodes at ambient topology is matched to a spatial formula node. Match processes at different type of spatial formula nodes

are different. Different match processes are introduced after auxiliary heuristic functions which are explained below.

Heuristic Functions. Heuristic functions are used at matching the Parallel composition (|) and Somewhere ($\diamond$) connectives. Former works try to match every alternative while searching a match for these connectives. In our proposed method, the number of these trials are reduced by the help of auxiliary heuristic functions. Some connectives of ambient logic called wildcard connectives match different kinds of ambient topology. These connectives are used for matching ambients of ambient topology which are not expressed in formulas. The constant **T** of the logic matches any ambient topology assigned to it. Negation connective of the logic can be seen as another kind of wildcard connective. Negations matches any ambient topology unless the sub formula of the negation matches this ambient topology. Another source of wildcard property is Somewhere connectives. The parallel process of the parent ambient are neglected when searching sublocations. So if the sublocation search is obtained by applying $\downarrow$ one or more times, the associated Somewhere connective gains a wildcard property. Function *wildcard* is a recursive function used for determining if a node of formula graphs has wildcard property.

It is not obvious to see which ambients are expected at sub formulas of Disjunction and Somewhere connectives. *guessExpectedAmbients* function is a recursive function which returns a set of expected ambient combinations for a formula graph node. The returned set includes all possible ambient combinations expected by children of that node. The returned value is a set instead of a single ambient combination. Function *findSublocation* is a recursive function used to find parent of an ambient at an ambient topology.

Matching of Spatial Formula. In a match between an ambient topology and spatial formula graph, all nodes of ambient topology must be matched with a node of spatial formula graph. Some nodes of spatial formula graphs can forward the ambient topology nodes assigned to them to their children, while others match assigned ambient topology nodes directly. The proposed spatial model checking algorithm tries alternative assignments of a given ambient topology nodes over a given spatial formula graph. The proposed spatial model checking algorithm is recursive where matching process starts from the roots of a graph and continues to underlying levels. If a suitable matching found at the upper level then matching process continues to find matches in lower levels. The match process is regulated by the semantics of spatial formula graph nodes.

4.4 Generation of Kripke Structure

A Kripke Structure is a state transition system where states are labeled by the set of atomic propositions which hold in that state. Atomic propositions can be considered as the marking of system properties.

Definition 4. *Let AP be a non-empty set of atomic propositions. A Kripke Structure is a four-tuple; M = (S, S0, R, L) where S is a finite set of states, S0*

$\subseteq S$ is the set of initial states, $R \subseteq S \times S$ is a transition relation, and $L: S \rightarrow 2^{AP}$ is a function that labels each state with the set of atomic propositions that are true in this state.

The state transition data structure provides sets S, S0 and relation R of a Kripke Structure. The elements of the set of atomic propositions come from formula reduction. In formula reduction, spatial formulas are replaced with atomic propositions. The function L is generated by applying spatial model checking for each state in state transition data structure against each spatial formula. Kripke Structure is obtained by attaching the values, coming from spatial model checking, into the state transition system graph.

4.5 NuSMV Code Generation

The model checking mechanism explained above provides CTL formulas and a Kripke Structure. The next step is the generation of NuSMV code which is semantically equivalent to the Kripke Structure and temporal logic formula. In the NuSMV specification a variable *state* is used for specifying states in the Kripke Structure. The other kind of variables used in NuSMV code generation is boolean variables for representing atomic propositions. CTL formulas provided by the formula reduction step are then converted to NuSMV code according to CTL formula graph provided by formula reduction, where the Sometime ($\Diamond$) connective is represented as EF and Everytime ($\Box$) connective is represented as AG. The atomic propositions are reflected into strings with their names.

4.6 Example for Spatial Model Checking Algorithm

Let's consider the scenario and policy example presented in Section 3.3. When the Ambient Calculus specification is input to the model checker, a total of 53 states are generated. One Atomic Proposition (AP) is generated, where

$$AP = \diamond\{\diamond Host2[\diamond\{Data1[T]|Data2[T]\}]|T\} \tag{4}$$

A part of the execution of the algorithm is presented in Table 3. Only the initial and the last two states are shown. For each state an action is executed to produce a new spatial state. For state 53 the spatial model checking algorithm matches the spatial formula AP to the current state of World.

Table 3. Part of output generated by the spatial model checker for the example policy presented in 3.3

State	Spatial state of World	AP	Action
0	Domain1[User1[]||Host1[File1[Data1[]]]]| Domain2[Host2[User2[]||File2[Data2[]]]]	F	User2[out Host2]
52	Domain1 [User1 [Host1 [File1[]]]| Domain2 [Host2 [File2 [User2[]||Data2 []||Data1 []]]	F	User2[out File2]
53	Domain1 [User1 [Host1 [File1[]]]| Domain2 [Host2 [File2 [Data2 []||Data1 []]||User2[]]	T	-

4.7 Complexity and Performance Analysis

Time Complexity. Time complexity of generation states transition system is dependent on the number of capabilities. The execution of a capability causes a future state. In the worst case, all capabilities are independent. Independence of capabilities means that capabilities are in sequence or they operate on different ambients. Where n is the number of capabilities in the ambient calculus specification, the time complexity of generating state transition system in worst case is

$$\sum_{k=0}^{n} \frac{n!}{k!} \tag{5}$$

The time complexity of checking spatial modalities are dependent to the type and number of the connectives of the spatial formulas. The overall time cost of the match process for Somewhere connective is linear with the cost of match process of parallel composition for specifications with Somewhere connectives. However, the time complexity of the match process is exponential with the number of ambients as defined in Formula 6 where a_{ne} is the number of topmost ambients of the ambient topology which are not expected by the heuristic functions, d_w is the number of disjunctions which have wildcard property in the parallel composition, *not* is the number of negations in the parallel composition, sw_w is the number of Somewhere connectives which have wildcard property in the parallel composition:

$$O(a_{ne}^{(sw_w+not+d_w)}) \tag{6}$$

In contrast, when the brute force search is used for decomposing ambient calculus specifications, the time complexity is calculated as defined in Formula 7 where $a = a_{ne} + a_e$ is the total number of topmost ambients in the ambient topology, including those expected by the heuristic functions (a_e), l is the number of location in the parallel composition, sw is the number of Somewhere connectives which have not wildcard property in a parallel composition, d is the number of disjunctions which have not wildcard property in the parallel composition:

$$O((a)^{(sw_w+sw+l+not+d+d_w)}) \tag{7}$$

As presented above, the variables that effect the exponential complexity of the match process is significantly reduced by the proposed algorithm.

4.8 Space Complexity

Proposed algorithm builds a state transition system in a depth-first manner. The depth of the state transition system is at most equal to the number of capabilities. Therefore, the space complexity of the space generation is $O(n)$ where n is the number of capabilities. When checking spatial modalities, the space needed is equal to the size of the formula which is dependent on the number of connectives of the formula. Therefore, the space complexity of checking spatial modalities is $O(c)$, where c is the number of the connectives at formula.

4.9 Performance

Due to space limitations, the details of performance tests will not be presented. As a summary, our performance tests suggest that the state transition system generation cost outweighs the spatial model check for both time and space consumption. As an example to performance results, a specification with 16 ambients and 37 capabilities generates nearly 630,000 states with memory consumption under 8 MB and a time of under 300 seconds. The performance test has been run on an Intel G5 server with 2.93 GHz CPU and 10 GB memory.

5 Future Work and Conclusions

We presented a method and tool for the specification and verification of security policies of multi-domain mobile networks. The main focus of this method is location and mobility aspects of security policies. The basic elements of this method are predicate logic, ambient calculus and ambient logic. In this paper, model checking techniques are applied for verification of security policies and an ambient calculus model checker is presented.

The size of the state transition system is the most significant element at time and spatial cost of model checking. Number of states grows exponentially as capability number increase linearly. A partial order reduction might decrease the number of the states of the state transition system and reduce time consumption and size of generated NuSMV code. Investigating partial order reduction techniques for ambient calculus is a direction for our future work.

In our ongoing research we are developing tools for automatic extraction of formal process calculus specifications and logic formulas from security policy.In order to extract the Ambient Logic formula and specification from security policy, we are building a tool called "Formal Specification Generator". The tool will be based on analysis of scenarios depicting sequences of actions of system elements. These high-level actions are more suitable for our problem domain in contrast to Ambient Calculus primitives. Therefore our aim is to provide an automated means to translate high level policy and actions to formal calculus and logic specifications.

References

1. Becker, M., Fournet, C., Gordon, A.: Design and semantics of a decentralized authorization language. In: 20th IEEE Computer Security Foundations Symposium, pp. 3–15. IEEE Computer Society Press, Los Alamitos (2007)
2. Bertino, E., Ferrari, E., Buccafurri, F.: A logical framework for reasoning on data access control policies. In: 12th IEEE Computer Security Foundations Workshop, pp. 175–189. IEEE Computer Society Press, Los Alamitos (1999)
3. Cardelli, L., Gordon, A.D.: Anytime, anywhere: modal logics for mobile ambients. In: 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL 2000, pp. 365–377 (2000)

4. Cardelli, L., Gordon, A.D.: Mobile ambients. Theoretical Computer Science 240(1), 177–213 (2000)
5. Charatonik, W., Gordon, A., Talbot, J.: Finite-control mobile ambients. In: Le Métayer, D. (ed.) ESOP 2002. LNCS, vol. 2305, pp. 295–313. Springer, Heidelberg (2002)
6. Charatonik, W., Zilio, S.D., Gordon, A.D., Mukhopadhyay, S., Talbot, J.: Model checking mobile ambients. Theoretical Computer Science 308(1-3), 277–331 (2003)
7. Compagnoni, A., Bidinger, P.: Role-based access control for boxed ambients. Theoretical Computer Science 398(1-3), 203–216 (2008)
8. Cuppens, F., Saurel, C.: Specifying a security policy: a case study. In: 9th IEEE Computer Security Foundations Workshop, pp. 123–134. IEEE Computer Society Press, Los Alamitos (1996)
9. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder policy specification language. In: Sloman, M., Lobo, J., Lupu, E.C. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 18–38. Springer, Heidelberg (2001)
10. Drouineaud, M., Bortin, M., Torrini, P., Sohr, K.: A first step towards formal verification of security policy properties for RBAC. In: Proc. QSIC (2004)
11. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. 4(3), 224–274 (2001)
12. Giunchiglia, C.C., Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: Nusmv: a new symbolic model checker. International Journal on Software Tools for Technology Transfer 4, 410–425 (2000)
13. Jajodia, S., Samarati, P., Subrahmanian, V.S.: A logical language for expressing authorizations. In: IEEE Symposium on Security and Privacy, pp. 31–42 (1997)
14. Jajodia, S., Samarati, P., Sapino, M.L., Subrahmanian, V.S.: Flexible support for multiple access control policies. ACM Trans. Database Syst. 26(2), 214–260 (2001)
15. Mardare, R., Priami, C., Quaglia, P., Vagin, O.: Model checking biological systems described using ambient calculus. Computational Methods in Systems Biology, 85–103 (2005)
16. Ryutov, T., Neuman, C.: Representation and evaluation of security policies for distributed system services. In: DARPA Information Survivability Conference and Exposition, pp. 172–183 (2000)
17. Scott, D.: Abstracting application-level security policy for ubiquitous computing. Ph.D. thesis, University of Cambridge (2005)
18. Sohr, K., Drouineaud, M., Ahn, G., Gogolla, M.: Analyzing and managing Role-Based access control policies. IEEE Transactions on Knowledge and Data Engineering 20(7), 924–939 (2008)
19. Unal, D., Caglayan, M.U.: Theorem proving for modeling and conflict checking of authorization policies. In: Proc. ISCN (2006)
20. Woo, T.Y.C., Lam, S.S.: Authorizations in distributed systems: A new approach. Journal of Computer Security 2, 107–136 (1993)
21. Zhang, N., Guelev, D., Ryan, M.: Synthesising verified access control systems through model checking. Journal of Computer Security 16(1), 1–61 (2007)