12

A Data Model for a Multilevel Replicated X.500 Server

G. Grossman
Arca Systems, Inc.
8229 Boone Blvd., Suite 750
Vienna, VA 22182-2623
Phone: +1 (703) 734-5611
FAX: +1 (703) 790-0385
Email: grossman@arca.com

M. Schaefer
Arca Systems, Inc.
10320 Little Patuxent Parkway, Suite 1005
Columbia, MD 21044
Phone: +1 (410) 715-0500
FAX: +1 (410) 715-0713
Email: marv@arca.com

Abstract

This paper describes a model of an X.500 directory augmented with replication for the introduction of multiple sensitivity levels. Semantic tags are applied to attributes to support consistent polyinstantiation and cover stories. Secure state and initial state are defined for the model, as well as the semantics of the operations that change the state. The model demonstrates that a coherent set of semantics can be constructed for such a system that preserve intra-level consistency as well as multilevel access control can be constructed for such a system.

This paper is based on work funded by Rome Laboratories under contract number F30602-95-C-0191.

Keywords

Directory service, X.500, multilevel security, database replication, data model.

P. Samarati et al. (eds.), *Database Security*

1 INTRODUCTION

The data model for a multilevel X.500 directory described here demonstrates the feasibility of constructing consistency-preserving update semantics in a replicated multilevel environment where polyinstantiation of directory entry attributes is present. It constitutes a detailed semi-formal presentation of the informal data model that is introduced in the *Trusted MLS Directory Service Phase I Final Report.* [Arca96]. It presents an abstract representation of a polyinstantiated multilevel tree of directory entries and defines the semantics of the update operations on the tree that maintain its consistency.

Section 2 describes the system to be modeled. Section 3 identifies the set of problems to be solved by the model. Section 4 describes the model itself. Section 5 presents conclusions.

2 SYSTEM TO BE MODELED

A system that provides X.500 services is called a Directory Service Agent (DSA). (A full description of X.500 is beyond the scope of this paper. The reader is referred to [ISO9594-1] through [ISO9594-9] as well as [Stee93], [Chad94], and [Waug94].) As shown in Figure 1, a DSA

- implements the storage of directory entries in a Directory Information Tree (DIT);
- communicates with users (Directory User Agents or DUAs) via the Directory Access Protocol (DAP) so long as a Binding (an association between the user and the DSA) is established;
- communicates with other DSAs to implement directory distribution (storage of different parts of the DIT in different DSAs) via the Directory Service Protocol (DSP) so long as a Hierarchical Operational Binding (an association between DSAs for this purpose) is established; and
- communicates with other DSAs to implement directory replication (storage of copies of the same part of the DIT in different DSAs) via the Directory Information Shadowing Protocol (DISP) so long as a Shadow Operational Binding (an association between DSAs for this purpose) is established.

For a discussion of the issues involved in producing a multilevel X.500 DSA, see [Arca96].

The X.500 DIT consists of entries, each of which has a *relative distinguished name* (RDN), represented by A, B, C, and D in Figure 1, and a *distinguished name* (DN) which is its path to the root, represented by the path [..., R, B, C] for the entry C in the figure. (The figure shows a DSA which does not store the root of the DIT, but only a subtree with the local root R. The ellipsis represents the portion of the DN between the root of the DIT and R.) Each entry contains *attributes*, each of which has a *type* and a

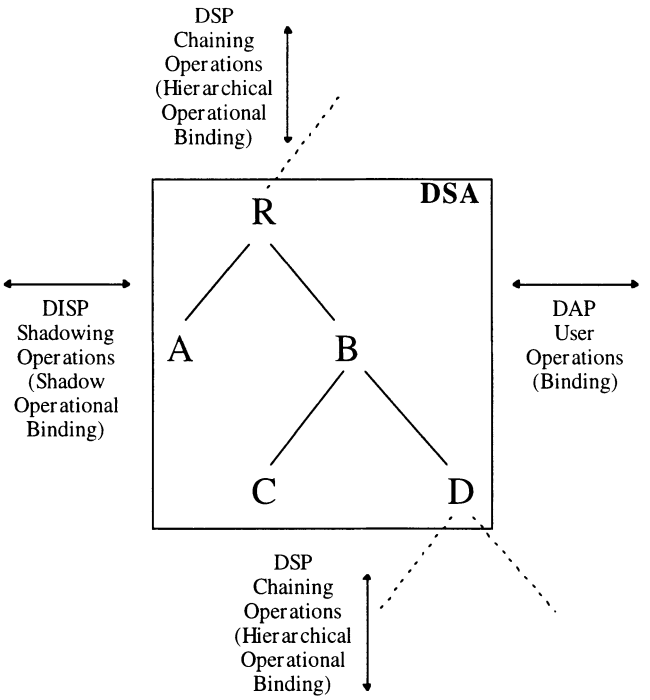

Figure 1 X.500 Directory Service Agent (DSA).

value. An attribute's type does not signify a data type, but rather the name by which the attribute can be denoted for retrieval or modification of its value.

To provide for data at multiple levels, the system to be modeled, shown in Figure 2, introduces multiple partitioned sensitivity level domains called *level-partitions*. Each two adjacent level-partitions are connected only via a SINTRA *Pump* [Kang94a] [Kang94b], which permits only updates from the lower of the two levels to flow to the higher. No information is allowed to flow in the other direction (except for acknowledgments from the higher to the lower of the two Pump levels, which are trusted not to leak to the lower level even the information that the acknowledgment has occurred).

3 PROBLEMS ADDRESSED BY THE MODEL

The model described here addresses the problems of labeling and Mandatory Access Control (MAC), including the Simple Security Property and the *-Property [BeLa76] that must be solved by any multilevel system. It also attempts to solve the problems of polyinstantiation [Denn87] and cover stories that are inherent in multilevel database management systems.

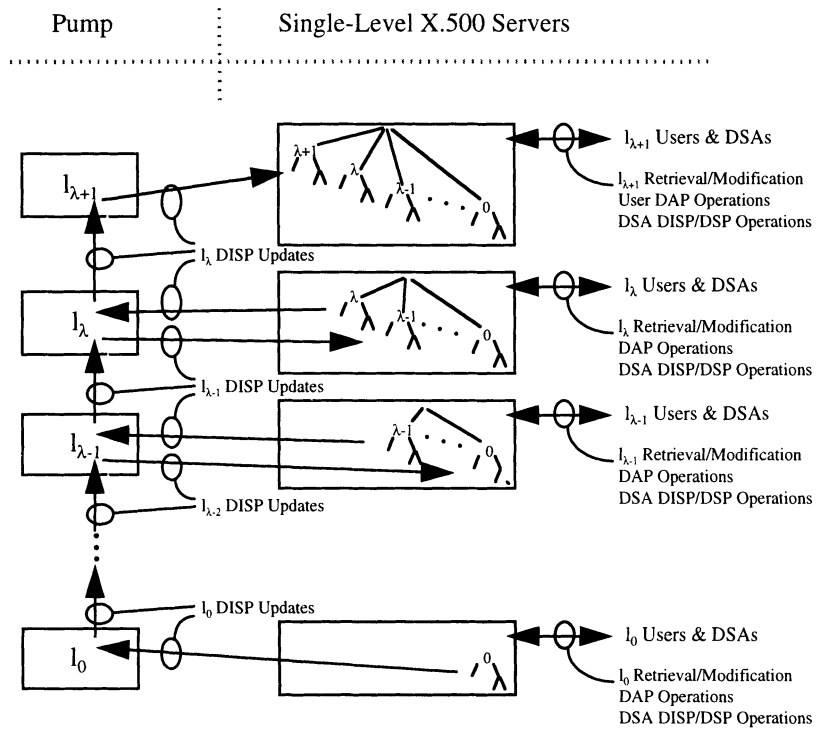

Figure 2 Multilevel X.500 DSA using replication.

In the system to be modeled, MAC is implemented architecturally. That is, only users logged-in (via the DAP *bind* operation) at a given level λ can access the level partition corresponding to λ (called the λ *level-partition*). Separation between the connections of the users at the various levels to the corresponding level partitions is maintained physically and/or cryptographically. The λ level-partition contains the information created at λ, as well as the information "pumped up" from each of the level-partitions whose levels are dominated by λ. (While the λ level-partition might contain an image of all of the information created at levels dominated by λ, this need not be so in the general case; X.500 replication is selective and it may not be necessary to see all lower-level information at higher levels.)

To maintain a consistent view for users at each level, the representation of the DIT in the λ level-partition (called the λ *tree*)is further partitioned into subtrees, one for each level represented in the system that is dominated by λ. The root of each of these subtrees is a node that defines the level of origin of the information in the subtree. A user at λ can modify at most the information in the λ subtree of the λ tree. (The X.500 standard defines discretionary access control mechanisms that may proscribe

modification of information even though it is modifiable under mandatory access control.)

If a user at λ wished to create or modify information concerning an entity already represented by information created at a level μ that is dominated by λ, the potential for polyinstantiation would exist. This would be represented by an entry e_λ in the λ subtree of the λ tree with the same DN (except for the substitution of λ for μ) as another entry e_μ in the μ subtree. These entries would have attributes a_λ in e_λ and a_μ in e_μ with the same types but with different values. The value of a_μ would constitute a "cover story" at μ for the more sensitive value of a_λ. In a simplification of the concept of semantic vectors introduced in the Trusted ONTOS Prototype [Scha95] to provide cover stories in the general ODBMS environment, the system to be modeled introduces a "semantic tag" mechanism: conceptually, there is a corresponding such tag to each attribute of each entry; if the value of the tag is **local**, then the value of the attribute is the value stored for the attribute in the present entry; otherwise, the value of the tag is **derived**, and the value of the attribute is the value of an attribute of the same type in some corresponding entry at a dominated level. The definition of the X.500 schema has been expanded to provide for selection at each level of the attributes for which cover stories can be constructed; an explicit special operation is required to create a cover story. The semantics defined in the model do not permit accidental polyinstantiation.

The goal of the model is to define the mechanisms of the system with sufficient detail and rigor that it is possible to

- define the secure states of the system
- define the initial state of the system and show that it is one of the secure states
- define the operations on the system and show that if each operation begins in a secure state, then it yields a secure state

The level of detail that is required to achieve this results in a model that is too large to be presented here in its entirety. We therefore present a high-level description of the model.

4 DESCRIPTION OF THE MODEL

The model first constructs the elements (entries, attributes, etc.), structure, and naming of the standard X.500 DIT. It then defines the structure of the level-partitions of the multilevel directory, including the notion of subjects, defines a subject's view as all of the entries whose levels are dominated by the subject's level, and shows that the set of grant of entries for which read access is granted for a subject is equivalent to the subject's view. The model then defines the states of the system and defines the secure states and the initial state. Finally, the model defines the operations on the system and the rules under which each operation yields a secure and consistent state.

5 THE DIRECTORY INFORMATION TREE (DIT)

5.1 Elements of the DIT

An X.500 directory consists of a tree of *entries*, each of which contains *attributes*. Each attribute has a *type* . The type governs the structure and function of the attribute. Each attribute also has a *value*, whose form is governed by the type of the attribute. Finally, each attribute has a *semantic tag*, which may have one of the values local or derived. The semantic tag governs the computation and modification of the attribute's value.

An attribute, then, is a triple consisting of a type, a value, and a semantic tag.

Each entry has a set of *classes* that define its structure and function. Each class is a set of attribute types.

An entry, then, is a set containing classes and attributes.

A class is *structural* if it defines the relationships between the entry and other entries in the tree. A subset of the attribute types in the structural class define the *name* of the entry. The name will be used to identify the entry (see **4.1.3**). Exactly one class in an entry must be a structural class.

An attribute may appear in an entry only if the attribute's type appears in a class that is a member of the entry. Each attribute type may appear in an entry no more than once.

The *schema* consists of the types and classes, together with the functions that define the structural classes and their name types.

5.2 Structure of the DIT

There is a distinguished entry called the *Root*, denoted by ρ. (In an implementation, it is likely that the Root will be the local root and the Administrative Point of an Autonomous Administrative Area, as well as the root of a Naming Context, but it could be the global root.)

Every entry e that is not the Root has associated with it a unique *superior* entry, designated $sup(e)$. The entries, organized under *sup*, form a tree called the DIT.

An entry e may have *subordinates*, for each of which e is the superior. Σ is the set of the immediate subordinates of the Root. It has a special function in naming entries and denoting their security levels.

The *inferiors* of an entry *e* are the elements of the tree with *e* as the root, but excluding *e*. No entry is its own inferior. (The DIT contains no loops.) All entries but the Root are contained in the inferiors of the Root. (All entries are in the DIT.)

The *subtree* of an entry *e* is the tree with *e* as the root, including *e*. Every entry not the Root is in the subtree (and the inferiors) of some other entry.

Every entry *e* not the Root is in the subtree of a single entry σ that is an immediate subordinate of the Root. σ is called the *subroot* of *e*.

The DIT, T, thus is defined by the set of entries *E*, the Root ρ, and the relation *sup*.

5.3 Names

The *Relative Distinguished Name* (*RDN*) of an entry is the set of attributes of the entry whose types are defined as the name types by the structural class of the entry. The RDNs for all subordinates of a given entry are distinct. The RDN is analogous to the primary key in a relational database, with each set of entry classes analogous to a relation.

The *Distinguished Name* (*DN*) of an entry not the Root is the RDN for the entry and, recursively, the DN for its superior, all the way back to a subordinate of the Root.

6 SENSITIVITY LEVELS

The set *L* of *sensitivity levels* consists of ordered pairs, one component of which is drawn from the set *H* of *hierarchical sensitivity levels* and the other from the set *N* of all of the sets in turn drawn from the set *K* of *non-hierarchical sensitivity categories*.

The sensitivity levels are partially ordered by the boolean function *dominates*.

7 THE MULTILEVEL DIRECTORY

The multilevel directory D consists of a set of sensitivity levels L and a set of level-partitions P.

7.1 Directory Levels

The set L denumerates the sensitivity levels present in D. The sensitivity level *sysHigh* dominates all other sensitivity levels in L. The set of sensitivity levels in D is ordered. (This restriction is not strictly necessary; it is made to simplify the description of semantic tags above and the derivation of attribute value below.)

7.2 Subjects and Clearance

To each element of the set S of subjects, there corresponds a sensitivity level given by the function *clearance*.

7.3 Level-Partitions

The set P consists of the set of level-partitions P_λ, one for each sensitivity level in L. Each level-partition consists of a level-tree, a level-schema, and level-subjects.

Level-Subjects

S_λ is the set of all subjects cleared at level λ.

Level-Schema

The schema defined above for the X.500 DIT is expanded into a level-schema for each level in the multilevel directory. To each level μ in L, there corresponds a set of types T_μ and a set of classes C_μ. The definition at each level is expanded to include a set K_λ of coverable types. A coverable type is one whose value at a dominated level can constitute a *cover story* for the value of the attribute at the present level. See Cover Stories, below.

Level-Tree

The DIT is expanded into a level-tree T_λ for each level. T_λ is called the λ-*tree*.

Level-Subtrees

Σ_λ is the set of all subordinates $\sigma_{\lambda\mu}$ of the root ρ_λ of the λ-tree. $\sigma_{\lambda\mu}$ we call the μ-*subroot in the* λ-*tree*, or, for brevity, the μ-*subroot in* λ. Similarly, we call the tree with $\sigma_{\lambda\mu}$ as its root the μ-*subtree in* λ.

Each entry that is a subordinate of the root has a *label* that is drawn from the set L.

In the λ-tree, each entry has a level which is: λ=*levelHigh* if the entry is the Root, equal to the entry's label if the entry is a subordinate of the Root, or equal to label of the subordinate of the Root which constitutes the root of the subtree of which the entry is a member.

From which it follows that all entries in a subtree of an entry that is a subordinate of the Root have the same label.

Counterparts

Two entries are *level-counterpart entries* if they are in level-subtrees at the same level in different level-trees and they have the same DN.

An entry e' is a *dominated-counterpart entry* of an entry e if they are in level-subtrees at different levels in the same level-tree, the level of e strictly dominates that of e', and they have the same DN.

An entry e' is a *near counterpart entry* to an entry e if e' is a dominated-counterpart entry to e and there is no dominated-counterpart entry to e whose level strictly dominates that of e.

An attribute a' of entry e' is the *nearest counterpart attribute* to an attribute a of entry e if e' is a near counterpart entry to e and the types of the attributes are equal.

If the semantic tag of an attribute a is derived, then there exists a nearest counterpart attribute to a.

Value Read from an Attribute

The value read from an attribute a is: if the semantic tag of a is derived, then (recursively) the value of the nearest counterpart attribute to a, otherwise the value of a.

Cover Stories

An attribute a of an entry e acts as a cover story for another attribute a' of an entry e' if e is a dominated counterpart of e', the types of the attributes are the same, and the semantic tag of a' is local.

8 VIEWS

The *view* of a subject s in S_λ is the set of all entries in subtrees of subordinates of the root of the λ-tree. In other words, the view of s is all of the entries in the λ level-partition, not including the root or its subordinates.

9 DISCRETIONARY ACCESS CONTROL

The X.500 specification provides an extensive Discretionary Access Control (DAC) scheme [ISO9594-3,4]. The model presented here focuses on mandatory aspect access control; each access control function is to be understood as conjoined with the appropriate X.500 DAC function.

10 READ ACCESS

All of the DAP interrogation operations (*read*, *compare*, *search*, *list*) eventually come down to reading information from an entry or from the attributes of the entry. For each

such request by subject s to read attribute a of the entry e, the read takes place if e is in the view of s.

11 STATES

The states of the multilevel directory and its elements are denoted by superscript non-negative integers. The order of the integers indicates the order of the states. D^t is thus the t^{th} state of the multilevel directory.

12 CREATION AND MODIFICATION

An entry is said to be *created* at a given state and level if such an entry does exist and if no such entry existed in the previous state at that level.

An attribute is said to be *modified* at a given state and level if such an attribute does exist and either

- no such attribute existed in the previous state at that level
- or if such an attribute did exist in the previous state, it had a different value.

13 SECURE STATE

State t is a secure state for the system if for each entry e in each level-partition of the state:

- if e is in the levelHigh-subtree of the partition, then e was created at that level
- otherwise there was a level-counterpart to e at some time in the past created at the level of the level-counterpart
- and
  - the level of e is the level of its subroot
  - and the level of the level-subtree containing e is dominated by the level of the level-partition
  - and for any attribute a of e, if a's semantic tag is local, then a was modified at the level of the level-subtree containing e.

14 INITIAL STATE

In the initial state of the system, all of the level-roots and level-subroots exist; these are the only entries.

15 STATE TRANSITIONS

The DAP and DSP operations that modify the state are *addEntry*, *deleteEntry*, *modifyEntry*, and *modifyDN*.

For a given level-partition, there are four sources for modifications to the DIT:

1. DAP operations (requests) coming directly from users at level
2. DSP chained operations (requests) coming from other DSAs at level and ultimately from users at level
3. DISP operations coming from peer DSAs at level, reflecting changes made to the DIT at level
4. DISP operations coming, via the Pump, from the next lowest level of the local DSA, reflecting changes to lower-level entries

Operations from sources 1 and 2 are ultimately *subject operations* and are so treated below. An additional subject operation, *permitCover*, has been added to enable the cover story mechanism.

Operations from source 3 presumably come from an authenticated trusted peer with which there is a Shadowing Agreement and a Shadow Operational Binding in effect. The protection and assurance features that are needed to trust such an association need further study. These are considered outside the model. Nonetheless, from a the point of view of policy at the level of the DIT entries, all Update Shadow operations will take place without decision at the DIT level. Thus these updates are also considered as occurring outside the model. Since these updates come from a master, they cannot conflict with user operations at this local DSA.

Operations from source 4 need to be tagged with the level of the source of the update to permit limiting the updates to the appropriate level-subtrees. These operations are called *Pumped Updates* below. Although the DISP operations are expressed differently from those of the other protocols, and many operations may come in one Update Shadow operation, such updates can be mapped onto the same four basic operations and will be so treated below.

15.1 Subject Operations

Subject modifyEntry Operation

A request by subject s to modify attribute a of the entry e is granted if the semantic tag of a is local, the clearance of s is equal to the level of e, and a is not in the RDN of e. (The RDN of an entry cannot be changed by this operation. The modifyDN operation must be used instead. This restriction was imposed by the X.500 designers so that name changes could be controlled by DAC separately from other modifications.)

Since the modification takes place at the highest level within the level-partition, no entries for which e is a counterpart need be modified within this level-partition to

reflect the modification of *e*. Each modification, however, must be propagated to the next level-partition via the Pump.

Subject deleteEntry Operation

A request by subject *s* to delete an entry *e* is granted if the clearance of *s* is equal to the level of *e* and *e* is not an immediate subordinate of the Root. (The level-defining subordinate roots cannot be deleted.)

Since the deletion takes place at the highest level within the level-partition, no entries for which *e* is a counterpart need be modified within this level-partition to reflect the deletion. The deletion must, however, be propagated to the nearest dominating level-partition via the Pump.

Subject addEntry Operation

A request by subject *s* to add an entry *e* immediately subordinate to an entry *e´* is granted if the clearance of *s* is equal to the level of *e´* and there is no subordinate of *e´* with an RDN equal to that of *e*. (It is necessary to ensure that the RDN, and hence the DN, of the new entry is unique.) The level of *e* is implicitly equal to the level of *s*.

The semantic tags for each of the attributes of the new entry are initialized to local. This "insulates" the values of the attributes of the new entry from any updates to counterpart entries at lower levels. Further, since the addition takes place at the highest level within the level-partition, no entries to which the new entry is a dominated counterpart need be created within this level-partition. The addition must, however, be propagated to the nearest dominating level-partition(s) via the Pump.

Subject modifyDN Operation

This operation can be used in two ways:

- to modify the RDN of an entry
- to modify the DN of an entry (move the entry within the DIT)

Handling the modification of the RDN for an entry appears straightforward, and is described below. In the context of multiple level-partitions and of labeling as a function of the DN, modification of the DN appears complex, requires further study, and is not discussed here.

A request by subject *s* to replace attribute *a* in the RDN of the entry *e* with the value *n* is granted if the semantic tag of *a* is local, the clearance of *s* is equal to the level of *e*, *e* is not an immediate subordinate of the root, and the new RDN of *e* would not be equal to that of another entry subordinate to *e´*.

Changing the RDN of *e* effectively deletes *e* and reinserts it with the same attribute values (except for attributes in the RDN) in the same structural place in the DIT. *e* can no longer be considered a representation of the same real-world entity as its lower-level counterparts (if any). If any attributes of e have derived semantic tags, the

values of these attributes must be copied from the nearest-dominated counterpart, and all such attributes marked local. This "insulates" *e* from any subsequently introduced counterpart entries at lower levels.

Since the modification takes place at the highest level within the level-partition, no entries for which *e* is a counterpart need be modified within this level-partition to reflect the modification of *e*. Each modification, however, must be propagated to the next level-partition via the Pump.

Subject permitCover Operation

A request by subject *s* to modify the semantic tag of the attribute *a* of the entry *e* from derived to local is granted if the semantic tag of *a* is derived, the clearance of *s* is equal to the level of *e*, and *a* is the set of coverable types for the level.

Since the modification of the semantic tag takes place at the highest level within the level-partition, no entries for which *e* is a counterpart need be modified within this level-partition to reflect the modification of *e*. Each modification, however, must be propagated to the next level-partition via the Pump. This operation has been added to support the cover story mechanism; it is not a defined X.500 operation. It, as well as the semantic tags themselves, could be implemented using standard X.500 constructs and operations.

15.2 Update Operations

Consistency checks must be performed on each update operation to ensure that an error in a lower-level untrusted back end in formulating the update information does not create an intra-level-partition inconsistency.

Pumped modifyEntry Update

Let *l′* be the level from which the update is pumped up. Then for each attribute *a* of an entry *e* to be modified it must be ensured that the semantic tag of *a* is local, the level of *l′* dominates the level of *e*, and *a* is not in the RDN of e.

Because of the definition of the semantic tags for the attributes of entries at levels above the level of the entry to be modified, no propagation of change within the current level-partition is necessary. The change must, however, be propagated to the nearest dominating level-partition via the Pump.

Pumped deleteEntry Update

Let *l′* be the level from which the update is pumped up. Then for each entry *e* to be deleted it must be ensured that the level of *l′* dominates the level of *e*.

Before *e* is deleted, the levels within the level-partition that dominate the level of *e* must be examined successively, from the nearest-dominating level up, to determine whether there exist any entries to which *e* is structurally a counterpart (i.e., an entry with the same DN except for the level-defining RDN). At each level:

- If such an entry, *e´*, exists at the dominating level, then examine the semantic tags of the attributes of *e´*.
  - If all of the semantic tags of all of the attributes of *e´* are derived, then delete *e´*. (It is an identical representation of the same real-world entity as is *e*.) Proceed to the nearest-dominating level.
  - If any of the semantic tags of any attributes of *e´* are local, then e´ is either a different representation of the same real-world entity as *e*, or a representation of another real-world entity. In either case, copy the values of all attributes of *e´* marked derived from *e*. Mark all such attributes local. This "insulates" *e´*, and any entries for which *e´* constitutes a dominated counterpart, from any subsequently introduced counterpart entries at lower levels. Stop.
- If no such entry exists at the dominating level, then stop. The entry at this dominating level has been deleted previously, and all previously-existing entries that dominate the previously-deleted entry have already been appropriately handled by this same procedure.

Delete *e*. All changes must be propagated to the nearest dominating level-partition via the Pump.

Pumped addEntry Update

Let *l´* be the level from which the update is pumped up. Then for each entry *e* to be added, immediately subordinate to an entry *e´*, it must be ensured that the level of *l´* dominates the level of *e* and that there is no subordinate of *e´* with an RDN equal to that of *e*.

As for the levels within the level-partition that dominate the level of the new entry, they must be examined successively, from the nearest-dominating level up, to determine whether there exists an entry to which the new entry is structurally a counterpart (i.e., an entry with the same DN except for the level-defining RDN). At each level:

- If no such entry exists at the dominating level, then create an entry with the appropriate RDN, each attribute of which is marked as derived.
- If such an entry, *e´*, exists at the dominating level, then stop. *e´* will have been created at the dominating level with all of its attributes marked as local, which effectively "insulates" *e´* from the (lower-level) versions of *e*. This creates a case of PolyLow-induced entry polyinstantiation.

All changes must be propagated to the nearest dominating level-partition via the Pump.

Pumped modifyDN Update

Let *l´* be the level from which the update is pumped up. Then for each attribute *a* in the RDN of an entry *e* to be modified it must be ensured that the semantic tag of *a* is local, *l´* dominates the level of *e*, *e* is not an immediate subordinate of the root, and the new RDN of *e* would not be equal to that of another entry subordinate to *e´*

Changing the RDN of *e* effectively deletes *e* and reinserts it with the same attribute values (except for attributes in the RDN) in the same structural place in the DIT. *e* can no longer be considered a representation of the same real-world entity as its lower-level counterparts (if any). If any attributes of e have derived semantic tags, the values of these attributes must be copied from the nearest-dominated counterpart, and all such attributes marked local. This "insulates" *e* from any subsequently introduced counterpart entries at lower levels.

The levels within the level-partition that dominate the level of *e* must be examined successively, from the nearest-dominating level up, to determine whether there are any name conflicts and whether there exist any entries to which *e* was structurally a counterpart before its name was changed (i.e., an entry with the same DN except for the level-defining RDN). At each level:

- If an entry, *e´*, exists at the dominating level with a conflicting name, then stop. *e* and *e´* do not represent the same real-world entity. *e´* will have been created at the dominating level with all of its attributes marked as local, which effectively "insulates" *e´* from the (lower-level) versions of *e*. This creates a case of PolyLow-induced entry polyinstantiation.
- If an entry, *e´´*, exists at the dominating level with the old name, and with all of its attributes marked as local, then stop. *e* and *e´* do not represent the same real-world entity. The marking of all attributes as local effectively "insulates" *e´* from the (lower-level) versions of *e*. This creates a case of PolyLow-induced entry polyinstantiation.
- If no entry exists at the dominating level with the old name, then stop. The entry at this dominating level has been deleted previously, and all previously-existing entries that dominate the previously-deleted entry will have already been appropriately handled.
- If an entry, *e´´´*, exists at the dominating level with the old name, and if the semantic tags of any of the attributes of *e´´´* are derived, then change the name of *e´´´*. (It is a representation of the same real-world entity as is *e*.) Proceed to the nearest-dominating level.

All changes must be propagated to the nearest dominating level-partition via the Pump.

16 CONCLUSION

The exposition above describes a model of an X.500 directory augmented with replication for the introduction of multiple sensitivity levels and providing at each level the replication of all lower level data, together with semantic tags on attributes to support consistent polyinstantiation and cover stories. Secure state and initial state are defined for the model, as well as the semantics of the operations that change the state.

BIBLIOGRAPHY

[Arca96] Arca Systems, *Trusted MLS Directory Service Phase I Final Report*, January 1996.

[BeLa76] Bell, D.E. and La Padula, L.J., *Secure Computer System: Unified Exposition and Multics Interpretation*, MTR-2997 Rev. 1, The MITRE Corporation, Bedford MA, 1976.

[Chad94] Chadwick, D., *Understanding X.500: The Directory,* Chapman and Hall, 1994.

[Denn87] Denning, D.E., Lunt, T.F., Schell, R.R., Heckman, M., and Schockley, W.R., "A Multilevel Relational Data Model", *Proceedings of the IEEE Symposium on Security and Privacy.* April 1987.

[ISO9594-1] ISO/IEC 9594-1, *Information technology–Open Systems Interconnection – The Directory: Overview of concepts, models and services*, 1995.

[ISO9594-2] ISO/IEC 9594-2, *Information technology–Open Systems Interconnection –The Directory: Models*, 1995.

[ISO9594-3] ISO/IEC 9594-3, *Information technology–Open Systems Interconnection–The Directory: Abstract service definition*, 1995.

[ISO9594-4] ISO/IEC 9594-4, *Information technology–Open Systems Interconnection –The Directory: Procedures for distributed operation*, 1995.

[ISO9594-5] ISO/IEC 9594-5, *Information technology–Open Systems Interconnection–The Directory: Protocol specifications*, 1995.

[ISO9594-6] ISO/IEC 9594-6, *Information technology–Open Systems Interconnection–The Directory: Selected attribute types*, 1995.

[ISO9594-7] ISO/IEC 9594-7, *Information technology–Open Systems Interconnection –The Directory: Selected object classes*, 1995.

[ISO9594-8] ISO/IEC 9594-8, *Information technology–Open Systems Interconnection–The Directory: Authentication framework*, 1995.

[ISO9594-9] ISO/IEC 9594-9, *Information technology–Open Systems Interconnection–The Directory: Replication*, 1995.

[Kang94a] Kang, M.H., Froscher, J.N., McDermott, J.P., Costich, O.L., and Peyton, R. "Achieving database security through data replication:

the SINTRA prototype". *Proceedings of the 17th National Computer Security Conference*, Baltimore, MD, September 1994.

[Kang94b] Kang, M.H., Froscher, J.N., McDermott, J.P., Costich, O.L., and Landwehr, C.E., "A Practical Approach to High Assurance Multilevel Secure Computing Service", *Proceedings of the 10th Annual Computer Security Applications Conference,* Orlando, FL, December 1994.

[Scha95] Schaefer, M., Martel, P., Kanawati, T., and Lyons, V., "Multilevel Data Model for Trusted ONTOS Prototype ", *Proceedings of IFIP WG 11.3 9th WC on Database Security.* Rensselaerville, NY, 13-16 August 1995.

[Stee93] Steedman, D., *X.500: The directory standard and its application*, Technology Appraisals, 1993.

[Waug94] Waugh, A., *X.500 and the 1993 Standard*, TR-SA-94-03, CSIRO Australia, March 1994.

Gary Grossman is Vice President, Research and Development at Arca Systems. He has thirty years of industry experience in security, networking, and operating systems, including product planning, definition, specification, design, and implementation, and the study of security evaluation/certification evidence. He has provided technical and management leadership on a number of innovative security projects, and has contributed to the design and definition of network protocols for the ARPANET, Internet, DoD, CCITT, and ISO. Mr. Grossman led the Arca effort to produce a multilevel replicated X.500 directory service.

Marv Schaefer is Chief Scientist at Arca Systems. Mr. Schaefer has more than 30 years experience on the theoretical foundations, design, implementation, and evaluation of trusted information systems. He provides technical leadership to senior professionals in Government and industry. Mr. Schaefer provides senior leadership and oversight on Arca's corporate efforts in information security, provides high-level advice and direction to government INFOSEC research and development organizations, and serves on government panels and working groups to address growing threat of Internet intrusion. Mr. Schaefer led the Trusted ONTOS data model effort, and supported the X.500 directory service effort.