Chapter 4

ROLE DELEGATION FOR A RESOURCE-BASED SECURITY MODEL

M. Liebrand, H. Ellis, C. Phillips, S. Demurjian, T.C. Ting and J. Ellis

Abstract Corporations and government agencies rely on inter-operating legacy, COTs, databases, clients, servers, etc., with security addressed from different dimensions. One dimension is delegation, where an authorized individual may delegate all or part of his/her authority to another individual, increasing security risk. This paper explores the inclusion of role delegation into a unified security model/enforcement framework that controls access to software APIs to limit, by role, which users can access which parts of APIs, constrained by time, classification (MAC), and data values. This paper examines role delegation, its incorporation into the security model, and, its impact on security assurance.

Keywords: Role delegation, delegation authority, resource-based security model

1. Introduction

The assembling of legacy, COTs, databases, clients, servers, etc., via middleware allows existing software artifacts to interoperate with new ones. The security capabilities must enforce the security policy for all users and protect sensitive information from access and misuse, controlling the access of individual users and their interactions. In the latter case, the delegation of authority, where an authorized individual (not the security officer) may delegate all or part of his/her authority to another individual, can increase security risk and impact on security assurance [2, 7, 8, 12]. This paper examines role delegation, proposing a means to allow individuals to delegate roles within security policy guidelines, while maintaining security run time assurance.

In support of this objective, we leverage our ongoing research on a unified, resource-based role-based/mandatory access control (RBAC/MAC) security model and enforcement framework for an environment comprised of software artifacts (interacting via middleware) [5, 9, 10], based on prior work [4]. Our approach concentrates on artifact APIs to support which portions of APIs

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35697-6_26

E. Gudes et al. (eds.), *Research Directions in Data and Applications Security*

can be invoked based on the responsibilities of a role, the security level of the user, and the values (parameters) and time of the invocation. Our enforcement framework, the Unified Security Resource (USR), is a resource with Security Policy, Security Authorization, and Security Registration services, augmented with the Security Policy Client (SPC) and the Security Authorization Client (SAC) administrative and policy maintenance tools [5, 9, 10].

The paper extends the RBAC/MAC security model and enforcement framework to support delegation at design time, and to incorporate delegation enforcement into the run-time environment. A secondary focus details the design and run time attainment of security assurance. To address these two foci: Section 2 presents a unified RBAC/MAC security model; Section 3 examines delegation extensions and analyzes our approach against various criteria [2]; Section 4 discusses security assurance for role delegation; and, Section 5 concludes the paper.

2. A RBAC/MAC Security Model

This section reviews the security model introduced in [5] and formalized in [9, 10]. We begin with lifetimes and sensitivity levels:

Definition 1: A *lifetime, LT,* is a time interval with start time (st) and end time (et), [st, et], et > st, and st/et is of the form (mo., day, year, hr., min., sec.). Concepts of LTs X and Y are: $X \triangleright Y$ means $Y.st \geq X.st$ and $Y.et \leq X.et$; $X \triangleleft Y \equiv Y \triangleright X$; If $ST = max\{X.st, Y.st\}$ and $ET = min\{X.et, Y.et\}$, then $Y \cap X$ is $\emptyset$ if $ET \leq ST$ or [ST, ET] if $ET > ST$; and $LT = [ct, \infty]$ is current time (ct) onward.

Definition 2: MAC concepts are: *Sensitivity levels*, $SLEVEL = \{U, C, S, T\}$ with unclassified (U), confidential (C), secret (S), and top secret (T), which form a hierarchy: $U < C < S < T$; *clearance (CLR)*, the SLEVEL given to users; and, *classification (CLS)*, the SLEVEL given to roles, resources, services, methods, etc.

Definitions 3 through 6 are for a distributed application of resources, services, and methods, with LTs (availability of resource/service/method) and CLSs (SLEVEL of a resource/service/method).

Definition 3: A *distributed application, DAPPL,* is composed of a set of unique *resources* (e.g., a legacy, COTS, DB, etc.), $R = \{R_i | i = 1..m\}$, each composed of a set of unique *services,* $S_i = \{S_{ij} | = 1..n_i\}$, each composed of a set of unique *methods,* $M_{ij} = \{M_{ijk} | k = 1..q_{ij}\}$.

Definition 4: Every *method* M_{ijk}, for $i = 1..m, j = 1, n_i, k = 1..q_{ij}$, of S_{ij} of R_i is registered as: $M_{ijk} = [M_{ijk}^{Name}, M_{ijk}^{LT}, M_{ijk}^{CLS}, M_{ijk}^{Params}]$ where M_{ijk}^{Name} is the name, M_{ijk}^{LT} is the LT when the method is available for use (default [ct, ∞]), $M_{ijk}^{CLS} \in SLEVEL$ (default U), and M_{ijk}^{Params} are parameter names and types.

Definition 5: Every *service* S_{ij}, for $i = 1..m, j = 1..n_i$, of R_i is registered as: $S_{ij} = [S_{ij}^{Name}, S_{ij}^{LT}, S_{ij}^{CLS}]$ where S_{ij}^{Name} is the name, S_{ij}^{LT} is the LT with $S_{ij}^{LT.st} \leq min\{M_{ijk}^{LT.st} | k = 1..q_{ij}\}$ and $S_{ij}^{LT.et} \geq max\{M_{ijk}^{LT.et} | k = 1..q_{ij}\}$, and $S_{ij}^{CLS} = min\{M_{ijk}^{CLS} | k = 1..q_{ij}\}$.

Definition 6: Every *resource* R_i, for $i = 1..m$, is registered as: $R_i = [R_i^{Name}, R_i^{LT}, R_i^{CLS}]$ where R_i^{Name} is the name, R_i^{LT} is the LT with $R_i^{LT.st} \leq min\{S_{ij}^{LT.st} | j = 1..n_i\}$ and $R_i^{LT.et} \geq max\{S_{ij}^{LT.et} | j = 1..n_i\}$, and$R_i^{CLS} = min\{S_{ij}^{CLS} | j = 1..n_i\}$.

Note that names, LTs, CLSs, etc., are set when the resource, services, and methods register with the unified security resource. For illustrative purposes, we use the U.S. Global Command and Control System (GCCS), that provides a commander with operational awareness of a crisis in near real-time. GCCS brings together 20 separate automated systems in over 625 locations worldwide [6]. Figure 1 contains a GCCS resource with two services, Joint and Component, with CLSs indicated.

```
Joint Service with Methods:
  (S)  Weather (Token);
  (S)  VideoTeleconference (Token, fromOrg, toOrg);
  (S)  JointOperationsPlannning (Token, CrisisNum);
  (S)  CrisisPicture (Token, CrisisNum, Grid1, Grid2);
  (S)  TransportationFlow (Token);
  (S)  LogisitcsPlanningTool (Token,  CrisisNum);
  (S)  DefenseMessageSystem (Token);
  (T)  NATOMessageSystem (Token);
Component Service with Methods:
  (S)  ArmyBattleCmdSys (Token, CrisisNum);
  (S)  AirForceBattleManagementSys (Token, CrisisNum);
  (S)  MarineCombatOpnsSys (Token, CrisisNum);
  (S)  NavyCommandSystem (Token, CrisisNum);
```

Figure 1. A GCCS resource with two services.

Definitions 7 through 18 involve the privilege specification process for user roles against resources, services, and methods.

Definition 7: A *user role, UR,* uniquely represents a set of responsibilities against a DAPPL, defined as: $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$ where UR^{Name} is the name, UR^{LT} is the LT (default [ct, ∞]), and $UR^{CLS} \in SLEVEL$ is CLS (default U).

Definition 8: *A user-role list,* $URL = \{UR_i | i = 1..r\}$, is the set of r unique roles for DAPPL, with each role as given in Def. 7.

Representative user roles for GCCS are shown in Figure 2, with the name, LT, and CLS given for each role. CDR_CR1 means commander of crisis 1, JPlanCR2 means joint planner of crisis 2, and so on. From a privilege perspec-

tive, URs will be granted access to resources, services, and methods, which have CLSs at or below the role's CLS. Using Figure 1, CDR_CR1 may be granted both services with JPlanCR1 given all methods from either service that have levels S, C, or U.

```
Roles: [CDR_CR1, [01dec00,01dec01],  T]
       [JPlanCR1, [01dec00, 01jun01], S]
       [JPlanCR2, [01jul01,01sep01], C]
       [ArmyLogCR1, [10dec00,01mar01], S]
       [ArmyLogCR2, [01jul01,01aug01], C]
Users: General DoBest:   [DoBest, [ct, infinity],  T]
       Colonel DoGood:   [DoGood, [01dec00,01jun01], T]
       Major DoRight:    [DoRight, [01dec00,01jan01], S]
       Major CanDoRight: [CanDoRight, [01jan01,01feb01, T]
URAs:  [JPlanCR1, CrisisPicture, [ct, infinity],true]
       [JPlanCR1, ArmyBattleCmdSys, [10dec00,16feb01], true]
       [ArmyLogCR1, CrisisPicture, [10dec00,16feb01],
                          Grid1 < NA20 AND Grid2 < NC40]
       [ArmyLogCR2, LogPlanningTool, [10dec00,16feb01],
                          CrisisNum=CR1]
```

Figure 2. Sample users, user-roles and user-role authorizations.

Definition 9: A *user, U,* is a unique entity accessing the DAPPL via a client application, and is defined as: $U = [U^{UserId}, U^{LT}, U^{CLR}]$ where U^{UserId} is the identifier, L^{LT} is the LT (default [ct,∞]), and $U^{CLR} \in SLEVEL$ is the clearance level (default U).

Definition 10: A *user list,* $UL = \{U_i | i = 1..u\}$, is the set of u users for DAPPL, where each user as given in Def. 9.

Representative users, Gen. DoGood, Col. DoBest, etc., for GCCS are shown in Figure 2, with the name, LT, and CLR given for each user.

For security assurance of users invoking methods by role, constraints on valid values, time limits, and CLR/CLS dependencies are utilized.

Definition 11: A *signature constraint, SC,* is a boolean expression on method M_{ijk}'s signature for $i = 1..m, j = 1..n, k = 1..g_{ij}$, of S_{ij} of a R_i, to limit the valid values on the parameters, M_{ijk}^{Params}.

For example, an ArmyLogCR1 UR can invoke method CrisisPicture (see Figure 1) limited by the SC ($Grid1 < NA20$ AND $Grid2 < NC40$). Thus, successful methods invocations are based on parameter values.

Definition 12: A *time constraint, TC,* is a LT, with default [ct,∞]), used as: UR and method LTs constrain the method assignment; UR, method, and user LTs constrain the method invocation; and UR and user LT constrain the user authorization to the role.

For example, the JPlanCR1 UR can have a TC of [10dec00, 16feb01] on the ArmyBattleCmdSys method of the Component service.

Definition 13: A *mandatory access control constraint, MACC,* is the *domination* of the SLEVEL of one entity over another. We check that the user's CLR $\geq$ role's CLS and that the role's CLS $\geq$ method's CLS.
MACC is used to compare CLR to CLS and deny or accept based on MAC rules [3]. MACC verifies if the user (with a CLR level) playing a role (with a CLS level) can invoke a method (with a CLS level) at ct.

The final definitions are for authorizations, namely, of method(s) to a user role, and of a user role to a user, to bring together all concepts.

Definition 14: A *user-role authorization, URA,* signifies that a UR is authorized to invoke a method at a particular time limited by certain values, defined as: $URA = [UR, M, TC, SC]$, with UR as given in Def. 7, M as given in Def. 4, TC as given in Def. 12 (default $[ct, \infty]$), and SC as given in Def. 11 (default true).
In Figure 2, URAs for JPlanCR1, ArmyLogCR1, and ArmyLogCR2 are shown, and illustrate the combinations of all earlier constructs.

Definition 15a: For each DAPPL, there is a *UR authorization matrix, URAM,* an $r \times q$ matrix, where $q = \sum_{i=1..m, j=1..n_i} q_{ij}$, indexed by roles and methods, with each entry defined as:

$$URAM(UR_i, M_j) = \begin{cases} 1 & UR_i \text{ is authorized to invoke } M_j \\ 0 & otherwise \end{cases}$$

Initially URAM contains all 0 entries. All entries with value 1 are *valid URA, VURA*. At design time, a URA must satisfy the CLS domination of role over method and the overlap of TC and LTs to become a VURA.

Definition 15b: A *valid user-role authorization list,* $VURAL = \{VURA_i \; \forall \; i = 1..v\}$, $v \leq r \times q$, is all VURAs (URAM(-,-) = 1).

Definition 16: A *user authorization, UA,* signifies that a user is authorized to play a specific UR, defined as: $UA = [U, UR, TC]$, with U as given in Def. 9, UR as given in Def. 7, and TC as given in Def. 12 for when the role is available to U (default $[ct, \infty]$).

Definition 17a: For each DAPPL, an $r \times u$ *user authorization matrix, UAM,* indexed by roles and users, is defined as:

$$UAM(UR_i, U_j) = \begin{cases} 1 & U_j \text{ is authorized to } UR_i \\ 0 & otherwise \end{cases}$$

Initially UAM contains all 0 entries. All entries with value 1 are *valid UA, VUA*. At design time, a UA must satisfy the CLR/CLS domination of user over role and the overlap of TC and LTs to become a VUA.

Definition 17b: A *valid user authorization list,* $VAUL = \{VUA_i | i = 1..w\}$, $w \leq r \times u$, is all VUAs (UAM(-,-) = 1).

Definition 18: A *client, C,* is an authorized user U, identified for a session by *client token* C = [U, UR, IP-Address, Client-Creation-Time].

Note that we will present additional examples of URs, users, URAM, and UAM, in Section 3.1, where they are inclusive of role delegation.

3. Role Delegation

Role delegation allows one user to transfer responsibility of a UR to another authorized user. In *administratively-directed delegation*, an infrastructure outside the direct control of a user mediates delegation [7]. In *user-directed delegation*, a user (playing a role) determines if and when to delegate responsibilities to another user to perform the role's permissions [8], with administrators continuing to establish the security policy and maintain delegation authority. Administration of RBAC, MAC, and delegation must be controlled to ensure that policy does not drift away from its original objective [11]. The remainder of this section investigates role delegation by: detailing extensions to our security model in Section 3.1; exploring issues of role delegation and revocation related to enforcement in Section 3.2; and analyzing our approach against a set of delegation criteria [2] in Section 3.3.

3.1 Model Extensions

In this section, we present the RBAC/MAC security model extensions for role delegation, to support changes to the enforcement framework (see Section 3.2) that attain security assurance (see Section 4).

Definition 19: A *delegatable* UR, DUR, is a $UR \in URL$ that is eligible for delegation.

Definition 20: The *delegatable UR vector, DURV,* is defined for all r $URs \in URL$ as:

$$DURV(UR_i) = \begin{cases} 1 & UR_i \text{ is a } DUR \\ 0 & UR_i \text{ is not a } DUR \end{cases}$$

Initially, DURV contains all 0 entries. As security requirements for a DAPPL are defined, the respective entries of DUR are set to 1. For the URs given in Figure 2, the roles CDR_CR1, JPlanCR1, JPlanCR2, are delegatable (resp. DURV(-) = 1) and ArmyLogCR1, and ArmyLogCR2 are not (resp. DURV(-) = 0).

Definition 21: An *original user,* $OU \in UL$, of a UR is authorized to the UR as part of the security policy (there exists a VUA for the OU/UR, i.e., UAM(UR,OU) = 1), and not as a result of a delegation.

Definition 22: A *delegated user,* $DU \in UL$, is a U that can be delegated a UR by an OU/DU (there is not a VUA for the DU/UR, i.e., $UAM(UR, DU) \neq 1$)), where a DU cannot be an OU for that same UR.

Definition 23: The *user delegation/authorization matrix, UDAM,* is indexed by roles and users, with each entry defined as:

$$UDAM(UR_i, U_j) = \begin{cases} 2 & UR_i \text{ is a DU of } UR_i \\ 1 & U_j \text{ is an OU of } UR_i \\ 0 & UR_j \text{ is not authorized to } DU_i \end{cases}$$

Initially UDAM contains all 0 entries. As users are authorized to roles via VUAs (Definition 17a), the relevant entries are set to 1 (see Figure 3).

```
UAM
User\User-Role  ArmyLogCR1 ArmyLogCR2 JPlanCR1 JPlanCR2 CDR_CR1
DoBest               0          0          0         0        1
DoGood               0          0          1         1        0
DoRight              1          0          0         0        0
CanDoRight           0          1          0         0        0
URAM
Method\User-Role  ArmyLogCR1 ArmyLogCR2 JPlanCR1 JPlanCR2 CDR_CR1
ArmyBattleCmdSys      1          1          1         1        1
CrisisPicture         1          1          1         1        1
MarineCombatOpnsSys   0          0          1         1        1
LogPlanningTool       1          1          0         0        1
UDAM
User\User-Role  ArmyLogCR1 ArmyLogCR2 JPlanCR1 JPlanCR2 CDR_CR1
DoBest               0          0          0         0        1
DoGood               0          0          1         1        0
DoRight              1          0          0         0        0
CanDoRight           0          1          0         0        0
DAM
User\User-Role ArmyLogCR1 ArmyLogCR2 JPlanCR1 JPlanCR2 CDR_CR1
DoBest               0          0          0         0        2
DoGood               0          0          1         1        0
DoRight              0          0          0         0        0
CanDoRight           0          0          0         0        0
```

Figure 3. Sample UAM, URAM, UDAM and URAM.

The remaining three definitions establish, for a given user of a role, the ability to have delegation authority (able to delegate) and/or pass-on delegation authority (able to pass on the authority to delegate).

Definition 24: *Delegation authority, DA*, is the authority given to the OU to allow delegation of a DUR to another user.

Definition 25: *Pass-on delegation authority, PODA,* is the authority given to an OU/DU to pass on DA for a DUR to another OU/DU.

Definition 26: The *delegation authority matrix, DAM,* is indexed by roles and users, with each entry defined as:

$$DAM(UR_i, U_j) = \begin{cases} 2 & U_j \text{ has DA and PODA for } UR_i \\ 1 & U_j \text{ has only DA for } UR_i \\ 0 & UR_j \text{ has neither DA nor PODA for } UR_i \end{cases}$$

Initially DAM contains all 0 entries; a sample DAM is in Figure 3.

To illustrate delegation, suppose that DoBest wishes to delegate (CDR_CR1) to DoGood with DA, with DoBest, CDR_CR1, and DoGood as given in Figure 2. This delegation can occur since DoBest is an OU (UDAM(CDR_CR1, DoBest) = 1) of CDR_CR1, DoGood is not an OU or DU (UDAM(CDR_CR1, DoGood) = 0), the UR is delegatable (assume DURV(CDR_CR1) = 1), and DoGood holds the correct clearance ((CDR_CR1 CLS = T) $\leq$ (DoGoodCLR= T)). DoBest can also grant DA because he has PODA (DAM(CDR_CR1, DoBest) = 2). Note that DoGood can execute the UR CDR_CR1, but is limited to his own LT. Also note that UAM(CDR_CR1, DoGood)=1, UDAM (CDR_CR1, DoGood)=1 (DU), DAM(CDR_CR1, DoGood)=1, (has DA) and VUA = [DoGood, CDR_CR1, [ct, ∞]] is created, as given in Figure 3.

3.2 Delegation/Revocation Rules

This section reviews extensions needed in support of our enforcement for role delegation. For example, if the OU delegates a UR, and then has that role revoked, the DU will also lose the delegated role. Thus, the relationships between users and delegated roles must be tracked, to maintain assurance and reduce risk or compromise of the security policy. The delegation and revocation rules for our enforcement framework are a simplified version of [12]. The two main differences are: 1. a simplification with DAM used to manage delegation depth; and 2. revocation only by the security officer. Our specific rules are: *User-To-User Delegation Authority Rule:* An OU/DU who is a member of a DUR, can delegate that UR to any user that meets the prerequisite conditions of the role: the DU receiving the role is not a member of the role; the OU or DU is identified as having delegation authority for the role; the DU meets the MACC; and the DU satisfies LT constraints. *Delegation Revocation Authorization Rule:* An OU can revoke any DU from a UR in which the OU executed the delegation. This is a stricter interpretation than [12], which allows any OU of a role revocation authority over a DU in the delegation path. In addition, a security administrator can revoke any delegation. *Cascading Revocation Rule:* Whenever an OU or DU in the delegation path is revoked, all DUs in the path are revoked.

3.3 Role Delegation Analysis

This section assesses our delegation work against a set criteria [2]:

Monotonicity (monotonic/non-monotonic) refers to the state of control the OU possesses after role delegation. We take a monotonic approach where the OU user does not relinquish control of the role, which offers a higher degree of control on delegation.

Permanent delegation is when a DU replaces the OU. *Temporary delegation* has a time limit with each UR. We have incorporated temporary delegation by allowing the OU to set LTs for each role delegation.

Totality refers to how completely the permissions assigned to the role are delegated. Partial delegation refers to the delegation of a subset of the permissions of the role. In total delegation, all of the permissions of the role are delegated, which we have chosen.

Levels of delegation refers to the ability of a DU to further delegate a role (PODA) and the number of vertical levels the delegated role can be delegated. Barka and Sandhu [2] identify three ways to control the depth of delegation: no control where roles can be re-delegated without limit; boolean control where roles can be re-delegated until a delegating user says no; and integer control where roles can be re-delegated until a certain number of re-delegations have occurred. We employ a modified boolean control and use DAM to control delegation by limiting the OU to DA and PODA which is consistent with our monotonicity approach.

Multiple delegations refers to the number of DUs (horizontally) to whom a DUR can be delegated to at any given time. We are including unlimited delegations in our security model since cardinality within a role has been found not to be used [1].

Cascading revocation is the indirect revocation of all DUs when the OU/administration revokes delegation of the OU's delegated role. Our enforcement framework supports cascading revocation since uncontrolled delegation is an unnecessary assurance risk.

Grant-dependency revocation refers to who has authority to revoke a DU. In grant-dependent revocation, only the OU can revoke the delegated role. In grant-independent revocation, any original member of the DUR can revoke a delegated role. We are utilizing a limited form of grant-independent revocation where only the delegating user and the security administrator can revoke a DUR.

4. Security Assurance

Security assurance checks must occur at both design and run times, so that the consistency of user roles, CLR/CLS levels, lifetimes, role delegations, and end-user authorizations, are verified when the security policy is changed. We provide assurance checks for delegation that augment the ones described in [10]. The intent of assurance checks is to have a DAPPL that provides strong confidence to system administrators and users in the attainment of the security policy with role delegation.

4.1 Design and Run Time Assurance

The management tools/enforcement framework for our security model and role delegation extensions must support security assurance for security officers creating the security policy, for users establishing delegation, and for enforcing the delegation/revocation rules (Section 3.2) and delegation concepts (Section 3.3). The first five assurance checks are for design and run times; the last two are only for run time. Note that we indicate the tool that supports the check at design time (the Security Authorization Client - SAC and the Security Delegation Client - SDC). The SDC is for delegation definition and revocation. We refer the reader to [13] for a discussion of all three tools.

MACC Domination: The user's CLR must dominate the UR's CLS. This check is needed so that MAC constraints are not violated during delegation. Checked by SAC tool.

Role Delegation: This check makes sure the DU is not already a member of the delegated role. For example, if user X is assigned role B, and is attempting to delegate B to user Y, then user Y cannot be an OU or a DU of role B (UDAM(B,Y) = 0). Checked by SAC tool.

User-To-User DA: An OU/DU who is a current member of a DUR can delegate that role to any user that meets the conditions: the receiving DU is not a member of the role; the OU/DU has DA for the role (DAM(UR,U) > 0); and, the DU meets MACC. Checked by SDC tool.

LT Consistency: In regard to permanence (Section 3.3), the LT of the DU is within the LT of the delegator (OU or DU), as given in Definition 21. Checked by SDC tool.

Modified Boolean Delegation: The combination of PODA and DA used in DAM controls the levels of delegation. An OU can delegate with DA or DA/PODA, but a DU can only have DA. This is enforced at design to limit to two levels. For a user to have PODA, UDAM(UR,U) = 1 (U is an OU) and DAM(UR,U) = 2 (U has DA and PODA) for the UR. Checked by SDC tool.

Delegation Revocation Authorization Rule: An OU can revoke any DU from a role in which the OU executed the delegation. In addition, a security administrator can revoke any delegation.

Cascading Revocation Rule: Whenever an OU or DU in the delegation path is revoked, all DUs in the path are revoked.

4.2 Security Assurance Rules

The assurance checks can be defined as *security assurance rules*, which are logical statements that must be satisfied for a status to be set (design time) or an action to be performed (run time). Rules I and II are for the assignment of DA and PODA to a user X at design time.

Rule I: Let $X \in UL$ be a user (Def. 9) and $A \in URL$ be a UR (Def. 7). X can have DA for A ($DAM(A,X) = 1$) iff $UDAM(A,X) = 1(X \text{ an } OU), DURV(A) = 1(A \text{ a } DUR)$, and $\exists$ a $VUA = [X, A, TC]$.

Rule II: Let $X \in UL$ be a user (Def. 9) and $A \in URL$ be a UR (Def. 7). X can have DA and PODA for A ($DAM(X,A) = 2$) iff $UDAM(A,X) = 1(X \text{ an } OU),\ DURV(A) = 1(A \text{ a } DUR)$, and $\exists$ a $VUA = [X, A, TC]$.

Rule III is for the delegation by a user of a role to another user, which sets UAM and UDAM for the delegated user and delegated role.

Rule III: Let $X \in UL$ be a user (Def. 9) and $A \in URL$ be a user role (Def. 7), such that $DAM(A,X) \geq 1$ (Rules I or II). X can delegate A to user Y limited by TC ($UAM(A,Y) = 1, UDAM(A,Y) = 2$, and $VUA = [Y, A, TC]$) iff $UDAM(A,Y) \neq 1, Y^{CLR} \geq A^{CLS}, TC = (Y^{LT} \cap A^{LT} \cap TC) \neq \emptyset$, and $TC.et > ct$. Set VUAL = VUAL $\cup$ UA=[Y, A, TC].

At run time, Rule IV is for DA/PODA and Rule V is for delegation.

Rule IV: Let $X \in UL$ be an OU or a DU, $A \in URL$ be a role, and $Y \in UL$ be a DU of A (Rule III is satisfied). Y can have DA for A (DAM(A,Y) = 1) if X has at least DA for A (DAM(A,X) $\geq$ 1). Y can have DA and PODA for A(DAM(A,Y)=2 if X has both DA and PODA for A (DAM(A,X)=2). (Rule IV limited to 2 levels in our framework).

Rule V: Let $X \in UL$ be an OU or a DU. Then X can delegate role A to DU Y limited by TC iff (Rule I or II) and Rule III.

5. Conclusions

This paper has examined the inclusion of role-delegation into a unified, resource-based RBAC/MAC security model and enforcement framework. To facilitate the discussion: Section 2 reviewed our security model [5, 9]; Section 3 presented role delegation as extensions to the security model, delegation rules to support enforcement [12], and, analysis against delegation criteria [2]; and, Section 4 explored security assurance checks and the accompanying rules to enforce assurance. Overall, we believe this work provides a strong foundation on role delegation for a resource-based setting. Please see [13] for full details on our work.

Acknowledgments

This research was supported in part by the GE Foundation through a grant to the University of Connecticut's School of Engineering.

References

[1] R. Awischus, Role-based access control with security administration manager, *Proceedings of the 2nd ACM Workshop on Role-Based Access Control*, 1997.

[2] E. Barka and R. Sandhu, Framework for role-based delegation models, *Proceedings of the 23rd National Information Systems Security Conference*, 2000.

[3] D. Bell and L. LaPadula, Secure computer systems: Mathematical foundations model, Technical Report M74-244, The Mitre Corporation, Bedford, Massachusetts, 1975.

[4] S. Demurjian and T.C. Ting, Towards a definitive paradigm for security in object-oriented systems and applications, *Journal of Computer Security*, vol. (4), 1997.

[5] S. Demurjian, et al., A user role-based security model for a distributed environment, *Research Advances in Database Information Systems Security*, Therrien (ed.), Kluwer, Dordrecht, The Netherlands, 2001.

[6] Joint Operational Support Center, http://gccs.disa.mil/gccs/, 1999.

[7] J. Linn and M. Nystrom, Attribute certification: An enabling technology for delegation and role-based control in distributed environments, *Proceedings of the 4th ACM Workshop on Role-Based Access Control*, 1999.

[8] S. Na and S. Cheon, Role delegation in role-based access control, *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, 2000.

[9] C. Phillips, et al., Security engineering for roles and resources in a distributed environment, *Proceedings of the 3rd International Systems Security Engineering Association Conference*, 2002.

[10] C. Phillips, et al., Towards information assurance in dynamic coalitions, *Proceedings of the 2002 IEEE SMC Information Assurance Workshop*, 2002.

[11] R. Sandhu and Q. Munawer, The ARBAC99 model for administration of roles, *Proceedings of the 15th Annual Computer Security Applications Conference*, 2000.

[12] L. Zhang, et al., A rule based framework for role-based delegation, *Proceedings of the 6th ACM Symposium on Access Control Models and Tools*, 2001.

[13] http://www.engr.uconn.edu/~steve/DSEC/dsec.html