Chapter 1

SAFETY AND LIVENESS FOR AN RBAC/ MAC SECURITY MODEL

C. Phillips, S. Demurjian, and T.C. Ting
Department of Electrical Engineering and Computer Science, U.S. Military Academy and Department of Computer Science and Engineering, University of Connecticut
charles.phillips@usma.edu and [steve, ting]@engr.uconn.edu

Abstract Our role-based/mandatory access control (RBAC/MAC) security model and enforcement framework for inter-operating legacy, COTS, GOTS, databases, servers, etc., limits: who (user/user role) can invoke which methods (based on value and MAC level) of artifact APIs at what times, and who (user) can delegate which responsibility (user role) at what times. In this chapter, we focus on assurance for the timeframe of access (of a user to a role, of a role to a method, etc.) and the attainment of the *Simple Security Property* (i.e., a subject cannot read information for which it is not cleared) and *Simple Integrity Property* (i.e., a subject is limited to writing information at its own level and lower), which together can be used to support *safety* and *liveness*.

Keywords: Security Assurance, Simple Security, Simple Integrity, Safety, Liveness.

1. Introduction

As agencies continue to assemble software artifacts (i.e., legacy, COTs, GOTS, databases, clients, servers, etc.), into interoperating applications, security assurance is critical [**?**, **?**, **?**, **?**]. In the area of mandatory access control (MAC), assurance-related properties are: *Simple Security Property*, a subject can read information at the same or lower clearance level [**?**]; *Strict *-Property*, a subject can only write information at exactly the level for which it is cleared [**?**]; *Liberal *-Property*, a subject with a low clearance level can write to an object with the same or higher clearance level [**?**, **?**]; and *Simple Integrity Property*, a subject can write information at its own level and lower [**?**]. In addition, assurance must

focus on application behavior with respect to its defined and realized security policy, to attain *safety* (nothing bad will happen to a subject or object during execution) [**?**] and *liveness* (all good things can happen to a subject or object during execution) [**?**].

In this chapter, we explore assurance for our role-based/mandatory access control (RBAC/MAC) security model and enforcement framework for interacting software artifacts [**?**, **?**, **?**]. Our approach focuses on which methods of APIs can be invoked based on the security classification level of a role, the security clearance level of the user, and the values (parameters), time, and classification level of the method. Assurance guarantees are provided to insure that: a user (with one lifetime) playing a role (with another lifetime) can invoke a method (yet another lifetime) at the current time; the method invocation does not violate MAC domination (*Simple Security* for read-only and read-write methods, and *Simple Integrity* for read-write methods); and, a degree of *safety* (nothing bads happens when a user invokes a method) and *liveness* (all good things happen when a user invokes a method) is attained.

In the remainder of this chapter, Section 2 defines our RBAC/MAC security model. Section 3 focuses on the formal proofs of the security assurance guarantees for the available timeframe of method invocation and satisfaction of Simple Security and Simple Integrity Properties, leading to the attainment of safety and liveness. Section 4 concludes the chapter.

2. A RBAC/MAC Security Model

This section reviews the security model [**?**, **?**] with delegation [**?**].

Def. 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.

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

For assurance, LTs can guarantee that definition and access to privileges will always satisfy time limits. MAC will be used to guarantee the Simple Security and Integrity Properties [**?**, **?**].

Defs. 3 to 6 are for a distributed application of resources, services, and methods, with LTs and CLSs.

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

Def. 4: Each *method* $M_{ijk} = [M_{ijk}^{Name}, M_{ijk}^{LT}, M_{ijk}^{CLS}, M_{ijk}^{Params}]$ for $i = 1..m, j = 1..n_i, k = 1..q_{ij}$, of S_{ij} of R_i has name M_{ijk}^{Name}, LT M_{ijk}^{LT} (default [ct, ∞]), $M_{ijk}^{CLS} \in SLEVEL$ (default U), and M_{ijk}^{Params} parameters.

Def. 5: Each *service* $S_{ij} = [S_{ij}^{Name}, S_{ij}^{LT}, S_{ij}^{CLS}]$ for $i = 1..m, j = 1..n_i$, of R_i has name S_{ij}^{Name}, LT $S_{ij}^{LT} = [min\{M_{ijk}^{LT.st}\}, max\{M_{ijk}^{LT.et}\}]$ $\forall k = 1..q_{ij}$, and $S_{ij}^{CLS} = min\{M_{ijk}^{CLS} | k = 1..q_{ij}\}$.

Def. 6: Each *resource* $R_i = [R_i^{Name}, R_i^{LT}, R_i^{CLS}]$ for $i = 1..m$, has name R_i^{Name}, LT $R_i^{LT} = [min\{S_{ij}^{LT.st}\}, max\{S_{ij}^{LT.et}\}]$ $\forall j = 1..n_i$, and $R_i^{CLS} = min\{S_{ij}^{CLS} | j = 1..n_i\}$.

For assurance, we guarantee the dependencies that exist among the LTs (resource LT spans its services' LTs; service LT spans its resources' LT), and the minimality of CLS (resource CLS is minimum of its services' CLS; service CLS is the minimum of its methods' CLS). For examples, we use the U.S. Global Command and Control System (GCCS), with Figure **??** containing Joint and Component services with (CLSs).

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

Def. 7: A *user role* $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$, represents a set of responsibilities, and has name UR^{Name}, LT UR^{LT} (default [ct, ∞]), and $UR^{CLS} \in SLEVEL$ (default U).

Def. 8: *A user-role list,* $URL = \{UR_i | i = 1..r\}$, has the r roles (Def. 7) for DAPPL.

For assurance, URs have specific LTs, used to check the "when" of access (i.e., can a user invoke a method at current time) and CLSs to check the "if" of access (i.e., does a UR's CLS dominate a method's CLS). For privileges, URs can be granted access to methods which have CLSs at or below the role's CLS (see Figure **??**).

Def. 9: A *user,* $U = [U^{UserId}, U^{LT}, U^{CLR}]$ is an entity accessing a client, and has a unique U^{UserId}, LT L^{LT} (default [ct,∞]), and $U^{CLR} \in SLEVEL$ (default U).

Def. 10: A *user list,* $UL = \{U_i | i = 1..u\}$, has the u users (Def. 9) for DAPPL.

For assurance, users have specific LTs for checking the "when" of access (i.e., can a user play a role at current time) and CLRs for checking the

```
(S) Joint Service      (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);
(S) Component Service (S)  ArmyBattleCmdSys (Token, CrisisNum);
                       (S)  AirForceBattleManagementSys (Token, CrisisNum);
                       (S)  MarineCombatOpnsSys (Token, CrisisNum);
                       (S)  NavyCommandSystem (Token, CrisisNum);
```

Figure 1.1. GCCS Joint and Component Services and their Methods.

```
Roles: [CDR_CR1, [01dec02,01dec03],  T]    [JPlanCR1, [01dec02, 01jun03], S]
       [JPlanCR2, [01jul01,01sep03], C]    [ArmyLogCR1, [10dec02,01mar03], S]
       [ArmyLogCR2, [01jul03,01aug03], C]
Users: General DoBest:   [DoBest, [ct, infinity],  T]
       Colonel DoGood:   [DoGood, [01dec02,01jun03], T]b
       Major DoRight:    [DoRight, [01dec02,01jan03], S]
       Major CanDoRight: [CanDoRight, [01jan03,01feb03, T]
URAs:  [JPlanCR1, CrisisPicture, [ct, infinity],true]
       [JPlanCR1, ArmyBattleCmdSys, [10dec02,16feb03], true]
       [ArmyLogCR1, CrisisPicture, [10dec02,16feb03], Grid1<NA20 AND Grid2<NC40]
       [ArmyLogCR2, LogPlanningTool, [10dec02,16feb03], CrisisNum=CR1]
```

Figure 1.2. Sample Users, User-Roles, and User-Role Authorizations.

"if" of access (i.e., does a user's CLR dominate a UR's CLS). Representative users for GCCS are shown in Figure **??**.

Def. 11: A *signature constraint, SC,* is a boolean expression on M_{ijk}^{Params}, for $i = 1..m, j = 1..n, k = 1..g_{ij}$, to limit values.

Def. 12: A *time constraint, TC,* is a LT (default [ct,∞]), and is utilized as follows: LTs of UR and method constrain the method assignment; LTs of UR, method, and user, constrain the method invocation; LTs of UR and user constrain the user authorization to the role; LTs constrain the time span of a delegation.

Def. 13: A *mandatory access control constraint, MACC,* is the *domination* of the SLEVEL of one entity over another, e.g., U's CLR $\geq$ UR's CLS or UR's CLS $\geq$ M's CLS.

For assurance, we provide guarantees that a user playing a role can invoke a method limited by parameter values (SC - Def. 11), method availability (TC - Def. 12), and the Simple Security and Integrity Properties applied by a user against a role invoking a method (Def. 13).

The next set of definitions is for authorizations of method(s) to user role(s) and of user role(s) to user(s), to bring together all of the concepts.

Def. 14: A *user-role authorization,* $URA = [UR, M, TC, SC]$, means UR (Def. 7) authorized to invoke M (Def. 4) within time TC

(Def. 12 - default $[ct, \infty]$), limited by values SC (Def. 11 - default true). Figure **??** has URAs for JPlanCR1, ArmyLogCR1, and ArmyLogCR2.

Def. 15a: A $r \times q$ *UR authorization matrix, URAM,* where $q = \sum_{i=1..m, j=1..n_i} q_{ij}$, is:

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

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

Def. 16: A *user authorization,* $UA = [U, UR, TC]$, means U (Def. 9) is authorized to play UR (Def. 7) for time TC (Def. 12 - default $[ct, \infty]$) for when the role is available to U.

Def. 17a: An $r \times u$ *user authorization matrix, UAM,* is:

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

Def. 17b: A *valid user authorization list*, $VUAL = \{VUA_i | i = 1..w\}$, $w \leq r \times u$, has all *valid UAs, VUAs* (Def. 17a).

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

For assurance, a valid URA (Defs. 14, 15a and 15b) can only be created if all required checks of a UR's capabilities against a M's characteristics are satisfied (i.e., TC and MACC checks), and the VURA then sets the criteria (UR, M, TC, SC) under which the invocation can occur. A corresponding set of guarantees also holds for a VUA.

The remaining set of definitions are for *role delegation* [**?**].

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

Def. 20: The *delegatable UR vector, DURV,* $\forall$ r $URs \in URL$ is:

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

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

Def. 22: A *delegated user*, $DU \in UL$, is a user 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 DU is not an OU for the same UR.

Def. 23: The $r \times u$ *user delegation/authorization matrix, UDAM,* is:

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

For assurance, there must be guarantees on the capabilities of an OU with respect to the a DU to insure that LTs/TCs and CLS/CLR are consistent; the delegation is defined and checked at runtime.

Def. 24: *Delegation authority, DA*, is given to OU for delegation of a DUR to another user.

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

Def. 26: The $r \times u$ *delegation authority matrix, DAM,* is:

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

Initially DAM has all 0 entries.

For assurance, there must be DA and PODA guarantees; DA allows an OU/DU to delegate, and PODA, allows an OU/DU to pass on that delegation authority to another DU, which is a dangerous privilege.

3. Towards Assurance Guarantees

Security assurance is critical in our model to allow the consistency of URs, CLR/CLS levels, LTs, role delegations, user authorizations, etc., to be verified when the security policy is defined, changed, and executed. Towards that end: Section 3.1 examines assurance guarantees on the *available time* of access for a method invocation; Section 3.2 details the attainment of the *Simple Security* and *Simple Integrity* Properties for method invocations; leading to Section 3.3 which explores *safety* and *liveness*. Overall, we have been influenced by others [**?**, **?**, **?**, **?**, **?**].

3.1 Time-Based Guarantees

To begin, we introduce *available time, AT*, which represents the maximum amount of time derived from various intersections of LTs and TCs of the authorizations for URs, OU, DU, and users, as given in Section 2. For example, $URA = [UR, M, TC, SC]$ involves LTs (for UR and M) and a TC, which must overlap to be a VURA. Formally, let $\Sigma^{LT} = \cap_{alli} \Sigma_i^{LT}$, where each Σ_i^{LT} represents a LT or a TC, using intersection for LTs/TCs as given in Def. 1. Then, any time t is in the maximum available time, AT, or $t \in \cap \Sigma_i^{LT} \forall\ i \ \Leftrightarrow\ t \in \Sigma^{LT}$. To assist in the proofs, we define:

Def. 1a: Let $Compare(X, Y)$ be a function that returns the overlap of LTs X and Y. $Compare(X, Y) = \{Y$ if $X \triangleright Y$; X if $Y \triangleright X$; $\emptyset$ if $Y \cap X = \emptyset$; $Y \cap X$ otherwise$\}$.

Using Def. 1a, we offer lemmas for the ATs for URAs (Def. 14), URs (Def. 7), UAs (Def. 16), DUs (Def. 22) and Users (Def. 9).

Lemma 1: If $URA_i = [UR, M, TC, SC]$ is a VURA, then $URA_i^{AT} = M^{LT} \cap UR^{LT} \cap TC$.

Proof:

1 By Defs. 14 and 15a, $URA_i = [UR, M, TC, SC]$ as a VURA means $URAM(UR, M) = 1$. By Def. 7, $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$. Set $URA_i^{AT} = UR^{LT}$.

2 By Def. 4, $M = [M^{Name}, M^{LT}, M^{CLS}, M^{Params}]$. Apply Def. 1a with $URA_i^{AT} = Compare(URA_i^{AT}, M^{LT})$.

3 By Def. 12, a TC is a LT. In $URA_i = [UR, M, TC, SC]$, TC constrains when UR authorized to M. Apply Def. 1a with $URA_i^{AT} = Compare(URA_i^{AT}, TC)$.

4 If $URA_i^{AT} \cap ct = \emptyset$, then $URA_i^{AT} = \emptyset$ and proof completes.

5 Otherwise, $URA_i^{AT} = M^{LT} \cap UR^{LT} \cap TC$ since it is equivalent to $URA_i^{AT} = URA_i^{AT} \cap TC$ (by steps 2 and 3 of the proof and substituting for URA_i^{AT} on the r.h.s. with $M^{LT} \cap UR^{LT}$). Note that if $URA_i^{AT} = \emptyset$, then there is no overlap.

Lemma 2: If $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$, then $UR^{AT} = \cap_{alli} URA_i^{AT}$, where each URA_i^{AT} is a VURA where $URAM(UR, M_j) = 1$ for some $j = 1..q$.

Proof:

1 Consider URAM as given in Def. 15a. If $URAM(UR, M_j) = 0$ for some j, then UR not authorized to M_j, and there is no need to use this entry to calculate UR^{AT}.

2 Consider all j where $URAM(UR, M_j) = 1$ (VURA of UR to M). If $\exists$ at least one j s.t. $URAM(UR, M_j) = 1$ and $M^{LT} = \emptyset$, then set $UR^{AT} = \emptyset$, and proof completes.

3 Let $URAM(UR, M_j) = 1$ and $URAM(UR, M_k) = 1$ for some j, k. Apply Def. 1a with $UR^{AT} = Compare(URA_{i_j}^{AT}, URA_{i_k}^{AT})$, where i_j/i_k is the VURA for M_j/M_k.

4 Repeat this step for each remaining entry $URAM(UR, M_j) = 1$, apply Def. 1a with $UR^{AT} = Compare(UR^{AT}, URA_{i_k}^{AT})$.

5 If $UR^{AT} \cap ct = \emptyset$, then $UR^{AT} = \emptyset$ and proof completes.

6 Otherwise, $UR^{AT} = \cap_{alli} URA_i^{AT}$.

Lemma 3: If $UA = [U, UR, TC]$ is a VUA, then $UA^{AT} = U^{LT} \cap UR^{AT} \cap TC$.

Proof:

1 By Defs. 16 and 17a, $UA = [U, UR, TC]$ as a VUA means $UAM(UR, U) = 1$. By Def. 9, $U = [U^{UserId}, U^{LT}, U^{CLR}]$. Set $UA^{AT} = U^{LT}$.

2 By Lemma 2, UR^{AT} is the AT of UR for all of its URAs. Apply Def. 1a with UA^{AT} =$Compare(UA^{AT}, UR^{AT})$. UA^{AT} represents the time that U can engage an authorized UR.

3 By Def. 12, a TC is a LT. For $UA = [U, UR, TC]$, apply Def. 1a with UA^{AT} =$Compare(UA^{AT}$,$TC)$.

4 If $UA^{AT} \cap ct = \emptyset$, then $UA^{AT} = \emptyset$ and proof completes.

5 Otherwise, $UA^{AT} = U^{LT} \cap UR^{AT} \cap TC$ since it is equivalent to $UA^{AT} = UA^{AT} \cap TC$ (by steps 2 and 3 of the proof and substituting for UA^{AT} on the r.h.s. with $U^{LT} \cap UR^{AT}$).

Lemma 4: If OU delegates one of its DUR, UR, to DU, then $DUA = [DU, UR, TC]$ has $DUA^{AT} = OUA^{AT} \cap DU^{LT}$.
Proof:

1 Let $OUA = [U, UR, TC]$ be a VUA for OU. By Lemma 3, OUA^{AT} is the AT for an OU to a UR. Set $DUA^{AT} = OUA^{AT}$.

2 For OU to delegate UR to DU, $UAM(UR, OU) = 1$. When OU delegates UR to DU, then $UAM(UR, DU)$ must be set to 1, and a $DUA = [DU, UR, TC]$ is created. The UR and TC in the DUA are as given in the OUA (Step 1), and thus do not impact DUA^{AT} since they are already in OUA^{AT}.

3 By Def. 9, $DU = [DU^{UserId}, DU^{LT}, DU^{CLR}]$. Apply Def. 1a with $DUA^{AT} = Compare(DUA^{AT}$,$DU^{LT})$.

4 If $DUA^{AT} \cap ct = \emptyset$, then $DUA^{AT} = \emptyset$ and proof completes.

5 $DUA^{AT} = OUA^{AT} \cap DU^{AT}$ since it is equivalent to $DUA^{AT} = DUA^{AT} \cap DU^{LT}$ (by Step 3 of the proof and substituting for DUA^{AT} on the r.h.s. with OUA^{AT}).

Lemma 5: If $U = [U^{UserId}, U^{LT}, U^{CLR}]$, then $U^{AT} = \cap_{alli} UA_i^{AT}$, where each UA_i^{AT} represents one VUA where $UAM(UR_j, U) = 1$ for some $j = 1..r$. This proof similar to Lemma 2 and omitted.

3.2 MAC Guarantees

Mandatory access controls are governed by strict rules for subjects (users) to access stored information or objects, and include many different properties: *Simple Security Property* [**?**], the *Strict *-Property* [**?**], the *Liberal *-Property* [**?**, **?**], and the *Simple Integrity Property* [**?**]. In this section, we prove that our RBAC/MAC model (see Section 2) satisfies both the Simple Security Property and the Simple Integrity Property as applied to methods, roles, and users. To do so, consider:

Def. 4a: Each $M_{ij} = \{M_{ijk} \mid k = 1..q_{ij}\}$, has read-only methods, $M_{ij}^{RO} = \{M_{ijk}^{RO}\ for\ some\ k\}$ and read-write methods, $M_{ij}^{RW} = \{M_{ijk}^{RW}\ for\ some\ k\}$; $M_{ij}^{RO} \cap M_{ij}^{RW} = \emptyset$ and $M_{ij}^{RO} \cup M_{ij}^{RW} = M_{ij}$.

Def. 28: Let $Dominate(X, Y)$ be a boolean function, with X and Y either CLR/CLS (Def. 2) with $U < C < S < T$. $Dominate(X, Y) = \{true\ if\ X \geq Y; false\ if\ X < Y\}$.

Given these definitions, we present a series of lemmas on URAs, URs, and UAs in regards to the Simple Security and Integrity Properties.

Lemma 6.1: If $URA_i = [UR, M, TC, SC]$ is a VURA, then URA_i satisfies the Simple Security Property for all M^{RO} and M^{RW}.
Proof by contradiction:

1 By Defs. 14 and 15a, $URA_i = [UR, M, TC, SC]$ as a VURA means $URAM(UR, M) = 1$. By Def. 7, $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$, where UR^{CLS} is the role's CLS. By Def. 4, $M = [M^{Name}, M^{LT}, M^{CLS}, M^{Params}]$, where M^{CLS} is the method's CLS. UR is the *subject* and M is the *object*.

2 If $M \in M_{ij}^{RO}$ or $M \in M_{ij}^{RW}$, it has a CLS, M^{CLS}. Suppose $Dominate(UR^{CLS}, M^{CLS}) = false$. Set $URAM(UR, M) = 0$ (Defs. 14, 15a, and 15b).

3 This contradicts Step 1 where $URAM(UR, M) = 1$.

4 Then, $Dominate(UR^{CLS}, M^{CLS}) = true$ ($UR^{CLS} \geq M^{CLS}$).

5 This is Simple Security (read-down/no-read-up) [**?**].

Lemma 6.2: If $URA_i = [UR, M, TC, SC]$ is a VURA, then URA_i satisfies the Simple Integrity Property for all M^{RW}.
Proof: Exact Steps as Lemma 6.1 except for Steps 2 and 6 (below).
2. If $M \in M_{ij}^{RW}$, it has a CLS, M^{CLS}. Suppose $Dominate(UR^{CLS}, M^{CLS}) = false$.
6. This is Simple Integrity (write-down/no-write-up) [**?**].

Lemma 6.3: Let $UR = [UR^{Name}, UR^{LT}, UR^{CLS}]$ and $\alpha = \cup_{all_i} URA_i$, where each URA_i is a VURA ($URAM(UR, M_j) = 1$ for some $j = 1..q$). If $Dominate(UR^{CLS}, M_j^{CLS}) = true$ for all $URA_i \in \alpha$, then UR satisfies the Simple Security Property for all methods and Simple Integrity Property for M^{RW}.
Proof:

1 For each $M \in M_{ij}^{RO}$ and $M \in M_{ij}^{RW}$, $Dominate(UR^{CLS}, M^{CLS}) = true$ (by Lemma 6.2), and, UR satisfies the Simple Security Property for all methods.

2 For each $M \in M_{ij}^{RW}$, $Dominate(UR^{CLS}, M^{CLS}) = true$ (by Lemma 6.2), and, UR satisfies Simple Integrity for all M^{RW}.

3 Steps 1 and 2 means that UR satisfies Simple Security for all methods and Simple integrity for M^{RW}.

Lemma 7: If $UA = [U, UR, TC]$ is a VUA, then UA satisfies the Simple Security and Integrity Properties for U CLR vs. UR CLS. Proof by contradiction:

1 By Lemma 6.3, UR satisfies Simple Security Property for all methods and the Simple Integrity Property for M^{RW}. Hence, $Dominate(UR, M_k) = true$ for all M_k authorized to UR.

2 By Defs. 16 and 17a, $UA = [U, UR, TC]$ as a VUA means $UAM(UR, U) = 1$. By Def. 9, $U = [U^{UserId}, U^{LT}, U^{CLR}]$, with U^{CLR}.

3 Suppose that $Dominate(U^{CLR}, UR^{CLS}) = false$.

4 Then, $UAM(UR, U) = 0$ (Defs. 16, 17a, and 17b), which contradicts Step 2.

5 Thus, $Dominate(U^{CLR}, UR^{CLS}) = true$ ($U^{CLR} \geq UR^{CLS}$).

6 Since UR satisfies both Simple Security and Integrity Properties, UA also does.

Lemma 8: If OU delegates UR in DUR to DU, then a delegated user authorization $DUA = [DU, UR, TC]$ satisfies both the Simple Security and Integrity Properties for delegated user (subject) vs. role (object). Proof similar to Lemma 7 and uses Lemma 4.

3.3 Safety and Liveness Guarantees

In this section, we explore *safety* (nothing bad will happen during a method invocation or delegation) [**?**] and *liveness* (all good can happen during a method invocation or delegation) [**?**] for our framework. We begin with a review of *security assurance rules, SARs* [**?**], which are logical statements that must be satisfied in order for a privilege to be set (design-time) or an action to be performed (runtime). The first four SARs are non-delegation checks at design time (Rule I for when an officer attempts to assign a method to a UR and Rule II for when an officer attempts to authorize a UR to a User) and runtime (Rule III for when a user selects a UR to play for a session and Rule IV for when a user attempts to invoke a method, via a client application).

Rule I: Let $A \in UR$ and M be a method. $URA = [A, M, TC, SC]$ is a VURA iff $A^{CLS} \geq M^{CLS}$, $TC = A^{LT} \cap M^{LT} \cap TC \neq \emptyset$, $TC.et > ct$; Set $URAM(A, M) = 1$, $VURAL = VURAL \cup URA$.

Rule II: Let $A \in UR$ and $X \in UL$. $UA = [X, A, TC]$ is a VUA iff $X^{CLR} \geq A^{CLS}$, $TC = X^{LT} \cap A^{LT} \cap TC \neq \emptyset$, $TC.et > ct$; Set $UAM(A, X) = 1$, $UDAM(A, X) = 1$, $VUAL = VUAL \cup UA$.

Rule III: Let $A \in UR$ and $X \in UL$. A can be authorized to X at runtime iff $UAM(A, X) = 1$ (Rule II is OK), and for the $VUA = [X, A, TC] \in VUAL$, $TC.et > ct$.

Rule IV: Let $A \in UR$, $X \in UL$, and M be a method. X with A can invoke M at runtime iff $UAM(A, X) = 1$ (Rule II is OK), $UDAM(A, M) = 1$ (Rule I is OK), for $VUA = [X, A, TC] \in VUAL$, $TC.et > ct$, for $VURA = [A, M, TC, SC] \in VURAL$, $TC.et > ct$, $SCOracle([M^{Name}, M^{Params}, M^{Values}], SC) = true$.

Note that in support of SCOracle, we assert the following: *There exists a SC Oracle that does not fail. The number and type of parameters vary with each method and SC will accept any combination. SC checks if the Boolean is satisfied, which has a very high expectation of success.*

Rules V and VI are for the assignment of DA and PODA to a user X. Rule VII is for design-time delegation of a UR by a user to another user. Rule VIII is for setting DA or DA/PODA from one user (either OU or DU) to another DU (Rule VII OK). Rule IX is for activating delegation.

Rule V: Let $X \in UL$ and $A \in URL$. X has DA for A ($DAM(A, X) =$ 1) iff $UDAM(A, X) = 1$(an OU), $DURV(A) = 1$(a DUR), and $\exists$ a $VUA = [X, A, TC]$.

Rule VI: Let $X \in UL$ and $A \in URL$. X can have DA and PODA for A ($DAM(X, A) = 2$) iff $UDAM(A, X) = 1$(an OU), $DURV(A) = 1$(a DUR), and $\exists$ a $VUA = [X, A, TC]$.

Rule VII: Let $X \in UL$ and $A \in URL$, s.t. $DAM(A, X) \geq 1$ (Rules V or VI). X can delegate A to user Y limited by 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 $UAM(A, Y) = 1, UDAM(A, Y) = 2$, and $VUAL = VUAL \cup UA = [Y, A, TC]$.

Rule VIII: Let $X \in UL$ be an OU/DU, $A \in URL$, and $Y \in UL$ be a DU of A. 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 VIII limited to 2 levels in our framework.

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

Now, we present theorems for the safety and liveness of Rules I to IV (delegation omitted). We define: *legal access* as an access authorized by a security policy; *liveness* to mean that every legal access is authorized [**?**]; and, *safety* to mean that no illegal access is authorized [**?**].

Theorem I: Rule I meets requirements for liveness and safety.

Proof:

1 Let A be a user role and M a method being authorized.

2 Initially, the URAM is set to all 0s. This is perfect access control as there are no UR authorizations to any methods, M_{ijk}, in this system. This meets safety (no bad things can happen), but not liveness, as no good things can happen.

3 Lemma 1 ensures that a URA is a VURA and $URA_i^{AT} = M^{LT} \cap UR^{LT} \cap TC$. If $URA_i^{AT} \neq \emptyset$ and $URA_i^{AT} \cap ct \neq \emptyset$, then M can be assigned to UR. Otherwise M cannot be assigned to UR because it can never be available.

4 Lemma 6.1 insures $A^{CLS} \geq M^{CLS}$.

5 To change an entry from 0 to 1 in a URAM, there must be a URA (Def. 14).

6 In order to build liveness, yet maintain safety, 0 entries in the URAM must be changed to 1 ($URAM(A, M) = 1$) for only those URs authorized to Ms (Defs. 15a and 15b). Validating changes to URAM allows for both safety and liveness; there is no access except for these specific authorizations and this is precisely Rule I.

Theorem II: Rule II meets requirements for liveness and safety.

Proof:

1 Let X be a user and A be the role being authorized.

2 Initially, the UAM and UDAM are set to all 0s. This is perfect access control as there are no users, U, authorized to any user roles, UR, in this system. This meets safety requirements, but not liveness, as no user can do anything.

3 U and UR must be available for access and $UR^{AT} \cap U^{AT} \cap ct \neq \emptyset$. Lemma 3 insures the maximum AT for a U to a UR. The intersection with the current time assures that available time has not passed, which is obviously a requirement for use.

4 Lemma 7 insures $X^{CLR} \geq A^{CLS}$.

5 To change an entry from 0 to 1 in UAM/UDAM, there must be a UA (Def. 16).

6 In order to build liveness, yet maintain safety, 0 entries in the UAM/UDAM must be changed. Validating changes to UAM and UDAM allows for both safety and liveness, as there is no access allowed except for these specific authorizations and this is precisely Rule II.

The safety/liveness of Theorems III/IV focuses on the runtime authorization of user to role, which is needed since the tranquility of CLR and CLS is not guaranteed, URs can change, and LTs and TCs can expire.

Theorem III: Rule III meets requirements for liveness and safety.
Proof:

1 Let X be a user and A be the role being authorized.
2 If $UAM(A, X) \neq 1$, deny runtime authorization. QED.
3 Revalidate Rule II: $X^{CLR} \geq A^{CLS}$, $TC = X^{LT} \cap A^{LT} \cap TC \neq \emptyset$, $TC.et > ct$. If $X^{CLR} < A^{CLS}$, then Rule II fails. If $TC = X^{LT} \cap A^{LT} \cap TC = \emptyset$, then Rule II fails. If either case is true, deny runtime authorization. QED.
4 Else, Rule II is revalidated at runtime, so by Theorem II, Rule II satisfies safety and liveness. Thus, Rule III does as well.

Theorem IV: Rule IV meets requirements for liveness and safety.
Proof:

1 Let X be a user, A its role, and M the method being called.
2 If $SCOracle([M^{Name}, M^{Params}, M^{Values}], SC) = false$, deny runtime authorization. QED.
3 If $UAM(A, X) \neq 1$, deny runtime authorization. QED.
4 If $URAM(A, M) \neq 1$, deny runtime authorization. QED.
5 If $U^{AT} = \emptyset$, (Lemma 3) deny runtime authorization. QED.
6 If $U^{TC.et} \leq ct$, deny runtime authorization. QED.
7 Otherwise, Rules I and II are revalidated enusring safety and liveness. Thus, Rule IV does as well.

Theorems V to IX correspond are structured in a similar fashion to prove safety and liveness for Rules V to IX

4. Conclusions

This chapter has examined security assurance guarantees for a RBAC and MAC security model and enforcement framework [**?**, **?**, **?**]. Section 2 reviewed our RBAC/MAC security model with delegation. Section 3 examined security guarantees with respect to available time (when an invocation can occur), security levels (authorizations of methods to user roles, and user roles to users), and for the attainment of safety and liveness (related to authorization, invocation, and delegation). The main contribution in this chapter is the series of lemmas and theorems that provide validation for the Simple Security and Simple Integrity Properties, which then lead to the stronger properties of insuring safety (nothing bad happens) and liveness (something good happens) as methods are invoked by users playing roles. Please see [**?**] for further information.

Acknowledgement: Thanks to Prof. C. Reynolds, USMA, Dept. of EE&CS for his input on the various formalisms/notation in Section 3.

References

[1] K. Alford, et al., "Information Assurance Pedagogy," *Proc. of IEEE Info. Assurance Wksp.*, 2001.

[2] B. Alpern and F. Schneider, "Defining Liveness," *Information Processing Letters*, Vol. 21, No. 4, 1985.

[3] D. Bell and L. LaPadula, "Secure Computer Systems: Mathematical Foundations Model," M74-244, Mitre Corp., Bedford, MA, 1975.

[4] E. Bertino, et al., "TRBAC: A Temporal Role-Based Access Control Model," *Proc. of 5th ACM Wksp. on RBAC*, 2000.

[5] K. J. Biba, "Integrity Considerations for Secure Computer Systems," TR-3153, Mitre Corp, Bedford, MA, 1977.

[6] S. Gavrila and J. Barkley, "Formal Specification For Role Based Access Control User/Role and Role/Role Relationship Management," *Proc. of the 3rd ACM Wksp. on RBAC*, 1998.

[7] Joint Operational Support Ctr., http://gccs.disa.mil/gccs/, 1999.

[8] L. Lamport, "Proving the Correctness of Multiprocess Programs," *IEEE Trans. on Software Engineering*, Vol. 3, No. 2, 1977.

[9] M. Liebrand, et al., "Role Delegation for a Resource-Based Security Model," *Data and Applications Security: Developments and Directions II*, E. Gudes and S. Shenoi (eds.), Kluwer, 2003.

[10] W. Maconachy, et al., "A Model for Information Assurance: An Integrated Approach," *Proc. of IEEE Info. Assurance Wksp.*, 2001.

[11] J. McCumber, "Information Systems Security: A Comprehensive Model," *Proc. of the 14th Natl. Computer Security Conf.*, 1991.

[12] S. Osborn, et al., "Configuring Role-Based Access Control to Enforce Mandatory And Discretionary Access Control Policies," *ACM Trans. on Information and System Security*, Vol. 3, No. 2, 2000.

[13] C. Phillips, et al., "Security Engineering for Roles and Resources in a Distributed Environment," *Proc. of 3rd ISSEA Conf.*, 2002.

[14] C. Phillips, et al., "Towards Information Assurance in Dynamic Coalitions," *Proc. of 2002 IEEE Info. Assurance Wksp.*, 2002.

[15] R. Sandhu, "Lattice-Based Access Control Models," *Computer Journal*, Vol. 26, No. 11, 1993.

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