Security II - Access Control Verification

Stefano Calzavara

Università Ca' Foscari Venezia

April 2, 2020

Università
Ca'Foscari
Venezia

Introduction

Along with authentication, authorization is the cornerstone of every secure system

- access control: grant or deny an access request performed by an authenticated subject
- what does it mean that an access control policy is secure?
- how can we prove (or disprove) security?

We will formally investigate this on a popular access control model.

Role-Based Access Control (RBAC)

The key idea of RBAC can be summarized as follows:

1. define a set of roles, i.e., collections of permissions
2. assign (sets of) roles to users, rather than individual permissions
3. access rights only depend on assigned roles, not on user identities
4. roles can be organized in a hierarchy: if $r_1 < r_2$, then r_2 inherits all the permissions of r_1

RBAC is very popular, because it greatly simplifies the assignment and revocation of permissions for large organizations.

RBAC: Example

Permission to Role Assignment

We assign permissions to roles as follows:

- Teacher: can create and grade assignments
- Teaching Assistant: can grade assignments
- Student: can submit solutions to assignments

User to Role Assignment

Stefano is a Teacher, all the other people are Students

Next year we will only need to revise the user-to-role assignment, which is great... but who is authorized to change the role assignment?

Administrative RBAC

ARBAC is the administrative extension of RBAC

- key idea: use RBAC to handle role assignment and role revocation
- a role r is said administrative if and only if it grants the ability to assign or revoke some role
- can-assign rules: express which roles can be assigned by owning r and under which conditions
- can-revoke rules: express which roles can be revoked by owning r and under which conditions

ARBAC: Example

Let's write down administrative rules in natural language:

1. a Teacher can revoke the role Student
2. a Teacher can revoke the role Teaching Assistant
3. a Teacher can assign the role Teaching Assistant to any user who is not a Student (conflict of interest)
4. a Teacher can assign the role Student to any user who is not also a Teacher (Teachers already spent enough time studying back then...)

Question Time!

Is this administrative policy secure or not?

Security of ARBAC

Some important observations:

- ARBAC is much harder to verify than RBAC, since it has a dynamic nature coming from the introduction of administrative rules
- there is no black-or-white notion of security for ARBAC: it really boils down to our security goals
- we need a way to formalize what are our security goals
- we need a way to prove whether the security goals are met or not

Modeling ARBAC

The ARBAC standard has three components:

1. URA: user-role administration, which deals with the common task of assigning and revoking roles to users
2. PRA: permission-role administration, which deals with the rare task of granting and removing permissions to roles
3. RRA: role-role administration, which deals with the uncommon task of changing the role hierarchy

We will focus just on the URA component of ARBAC, hence we do not model permissions and the role hierarchy.

ARBAC Semantics

We assume the existence of finite sets of roles R and users U

- we let $UR \subseteq U \times R$ represent a user-to-role assignment
- given a user $u \in U$, we let $UR(u) = \{r \in R \mid (u, r) \in UR\}$
- we let $\mathcal{P} = (CA, CR)$ stand for a policy including a set of can-assign rules CA and a set of can-revoke rules CR

State Transition System

Given an initial user-to-role assignment UR_0, the policy $\mathcal{P}$ induces a set of possible new user-to-role assignments UR_i reachable from UR_0 by applying the administrative rules of $\mathcal{P}$.

ARBAC Semantics

Can-Assign Rules

We let $CA \subseteq R \times 2^R \times 2^R \times R$, where each $(r_a, R_p, R_n, r_t) \in CA$ has the following meaning: users with administrative role r_a can assign role r_t to any user who has all the roles in R_p and none of the roles in R_n.

Can-Revoke Rules

We let $CR \subseteq R \times R$, where each $(r_a, r_t) \in CR$ has the following meaning: users with administrative role r_a can revoke role r_t from any user.

ARBAC Semantics

Given a policy $\mathcal{P} = (CA, CR)$, the transitions $UR_i \rightarrow_{\mathcal{P}} UR_{i+1}$ are defined by the following two rules:

$$\frac{(u_a, r_a) \in UR \qquad (r_a, R_p, R_n, r_t) \in CA \\ R_p \subseteq UR(u_t) \qquad R_n \cap UR(u_t) = \emptyset \qquad r_t \notin UR(u_t)}{UR \rightarrow_{\mathcal{P}} UR \cup \{(u_t, r_t)\}}$$

$$\frac{(u_a, r_a) \in UR \qquad (r_a, r_t) \in CR \qquad r_t \in UR(u_t)}{UR \rightarrow_{\mathcal{P}} UR \setminus \{(u_t, r_t)\}}$$

We omit the subscript $\mathcal{P}$ when it is irrelevant or clear from the context.

Security of ARBAC: Example

Let's translate the administrative rules in our framework:

1. a Teacher can revoke the role Student: $(T, S) \in CR$
2. a Teacher can revoke the role Teaching Assistant: $(T, TA) \in CR$
3. a Teacher can assign the role Teaching Assistant to any user who is not a Student: $(T, \emptyset, \{S\}, TA) \in CA$
4. a Teacher can assign the role Student to any user who is not also a Teacher: $(T, \emptyset, \{T\}, S) \in CA$

Question Time!

Can we formalize our security goals and reason about them?

Security of ARBAC: Example

We can show that, given a specific initial user-to-role assignment, our example policy leads to a conflict of interest: it is possible to have a user who is both a Student and a Teaching Assistant

$$\begin{array}{lll} \{(a,T),(b,S)\} & \rightarrow & \{(a,T)\} \\ & \rightarrow & \{(a,T),(b,TA)\} \\ & \rightarrow & \{(a,T),(b,TA),(b,S)\} \end{array}$$

Role Reachability Problem

The most useful problem to solve for the security verification of ARBAC is known as the role reachability problem.

Definition

Given an initial user-to-role assignment UR, a policy $\mathcal{P}$ and a role r_g, the role reachability problem amounts to checking whether there exists a user-to-role assignment UR' such that $UR \rightarrow_{\mathcal{P}}^{*} UR'$ and $r_g \in UR'(u)$ for some user u.

This property sounds very weak... why is it so useful?

Why Role Reachability?

Many useful problems can be reduced to role reachability.

Example (Mutual Exclusion)

Can the roles r_1 and r_2 be ever assigned together?

Encoding (let r_g be a fresh, unassigned role):

1. assign a fresh irrevocable role $\hat{r}$ to some user u
2. introduce a new can-assign rule $(\hat{r}, \{r_1, r_2\}, \emptyset, r_g)$
3. check role reachability for r_g
4. return the answer to point 3

Why Role Reachability?

Many useful problems can be reduced to role reachability.

Example (Bounded Safety)

Can the role r be assigned only to users $\{u_1, \ldots, u_k\}$?

Encoding (let r_g be a fresh, unassigned role):

1. assign a fresh irrevocable role $\hat{r}$ to users $u_1, \ldots, u_k$
2. introduce a new can-assign rule $(\hat{r}, \{r\}, \{\hat{r}\}, r_g)$
3. check role reachability for r_g
4. invert the answer to point 3

Why Role Reachability?

Many useful problems can be reduced to role reachability.

Example (Availability)

Will the role r be always assigned to user u?

Encoding (let r_g be a fresh, unassigned role):

1. assign a fresh irrevocable role $\hat{r}$ to user u
2. introduce a new can-assign rule $(\hat{r}, \{\hat{r}\}, \{r\}, r_g)$
3. check role reachability for r_g
4. invert the answer to point 3

Complexity of Role Reachability

The total number of possible user-to-role assignment is $O(2^{|R| \cdot |U|})$

- for $R = 3$ and $U = 30$, we have $2^{60} \approx 1.15 \times 10^{18}$ possibilities
- the computational complexity of the role reachability problem was proved to be PSPACE-complete [4]

How can we deal with this scary algorithmic complexity?

1. use restricted fragments of the ARBAC model
2. rely on approximated analysis techniques (false positives)
3. perform an aggressive pruning of the ARBAC policy

Restricted Fragments of ARBAC

Example (No Negation)

If the policy does not make use of negative preconditions, i.e., $R_n = \emptyset$ for all the can-assign rules, then the complexity class of the role reachability problem is P.

Example (No Revocation)

If the policy does not allow role revocation, i.e., $CR = \emptyset$, then the complexity class of the role reachability problem is still NP-complete.

More fragments and full formal details are available in [4].

Restricted Fragments of ARBAC

The separate administration property greatly simplifies the solution to the role reachability problem.

Definition

A policy $\mathcal{P} = (CA, CR)$ satifies the separate administration property if and only if the set of roles R can be partitioned in two sets AR, RR of administrative roles and regular roles respectively such that:

- for each $(r_a, R_p, R_n, r_t) \in CA$: $r_a \in AR$ and $R_p \cup R_n \cup \{r_t\} \subseteq RR$
- for each $(r_a, r_t) \in CR$: $r_a \in AR$ and $r_t \in RR$

Restricted Fragments of ARBAC

If a policy satifies the separation administration property, it is possible to modify $\mathcal{P} = (CA, CR)$ and the initial UR as follows:

1. identify the set $AR_0 = \{r \in AR \mid \exists u \in U : (u, r) \in UR\}$
2. revoke all the roles in $AR_0 \neq \emptyset$ from the users in UR
3. create a fresh user u_a (the administrator) with a fresh role r_a
4. replace the first component of all rules in $CA \cup CR$ with r_a
5. keep only a single user for each role combination in UR

This does not change the complexity, but greatly reduces R and U.

Restricted Fragments of ARBAC

Note that tracking just a single user for each role combination is unsound when the separate administration property does not hold!

- $CA = \{(r_1, \emptyset, \{r_1\}, r_2\}$
- $CR = \{(r_1, r_1)\}$
- $UR = \{(a, r_1), (b, r_1)\}$

The role r_2 is reachable here:

$$\{(a, r_1), (b, r_1)\} \rightarrow \{(a, r_1)\} \rightarrow \{(a, r_1), (b, r_2)\}$$

However, r_2 would not be reachable if we only kept (a, r_1) in UR.

Approximated Analyses of ARBAC

Approximated analyses can quickly return sound yet conservative results.

Example

Let $\hat{\mathcal{P}}$ stand for the policy obtained from $\mathcal{P}$ by removing all the negative preconditions of the can-assign rules. If r is not reachable in $\hat{\mathcal{P}}$, then it is not reachable in $\mathcal{P}$. This can be checked in polynomial time.

Two notable examples of approximated analyses for ARBAC:

- security types [1]
- program analysis [2]

Pruning Algorithms

Pruning algorithms can simplify instances of the role reachability problem by removing roles, users or administrative rules

- intuition: many roles, users and rules are useless for a specific instance of the role reachability problem
- building block of many other analyses as well
- state-of-the-art algorithm: aggressive pruning [3]

We will consider two simple algorithms here, called slicing algorithms.

Forward Slicing

Compute an over-approximation of the reachable roles:

- $S_0 = \{r \in R \mid \exists u \in U : (u, r) \in UR\}$
- $S_i = S_{i-1} \cup \{r_t \in R \mid (r_a, R_p, R_n, r_t) \in CA \wedge R_p \cup \{r_a\} \subseteq S_{i-1}\}$

Let S^* be the fixed point to this set of equations, then:

1. remove from CA all the rules that include any role in $R \setminus S^*$ in the positive preconditions or in the target
2. remove from CR all the rules that mention any role in $R \setminus S^*$
3. remove the roles $R \setminus S^*$ from the negative preconditions of all rules
4. delete the roles $R \setminus S^*$

Backward Slicing

Compute an over-approximation of the roles which are relevant to assign the goal of the role reachability problem:

- $S_0 = \{r_g\}$
- $S_i = S_{i-1} \cup \{R_p \cup R_n \cup \{r_a\} \mid (r_a, R_p, R_n, r_t) \in CA \wedge r_t \in S_{i-1}\}$

Let S^* be the fixed point to this set of equations, then:

1. remove from CA all the rules that assign a role in $R \setminus S^*$
2. remove from CR all the rules that revoke a role in $R \setminus S^*$
3. delete the roles $R \setminus S^*$

References I

Stefano Calzavara, Alvise Rabitti, and Michele Bugliesi.
Compositional typed analysis of ARBAC policies.
In *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 33–45. IEEE Computer Society, 2015.

Anna Lisa Ferrara, P. Madhusudan, and Gennaro Parlato.
Security analysis of role-based access control through program verification.
In *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*, pages 113–125. IEEE Computer Society, 2012.

References II

Anna Lisa Ferrara, P. Madhusudan, and Gennaro Parlato.
Policy analysis for self-administrated role-based access control.
In *Tools and Algorithms for the Construction and Analysis of System - 19th International Conference, TACAS 2013*, volume 7795 of *Lecture Notes in Computer Science*, pages 432–447. Springer, 2013.

Somesh Jha, Ninghui Li, Mahesh V. Tripunitara, Qihua Wang, and William H. Winsborough.
Towards formal verification of role-based access control policies.
IEEE Trans. Dependable Sec. Comput., 5(4):242–255, 2008.