Trust-based Dynamic RBAC

Tamir Lavi[1] and Ehud Gudes[1,2]
[1]*The Open University, Ra'anana, Israel*
[2]*Ben-Gurion University, Beer-Sheva, 84105, Israel*

Keywords: Trust-Based Access Control, RBAC, Privacy-Preservation, Role Delegation.

Abstract: A prominent feature of almost every computerized system is the presence of an access control module. The Role Based Access Control (RBAC) model is among the most popular in both academic research and in practice, within actual implementation of many applications and computer infrastructures. The RBAC model simplifies the way that a system administrator controls the assignment of permissions to individuals by assigning permissions to roles and roles to users. The growth in web applications which enable the access of world-wide and unknown users, expose these applications to various attacks. This led few researchers to suggest ways to incorporate trust within RBAC to achieve even better control over the assignment of users to roles, and permissions within roles, based on the user trust level. In this work, we present a new trust-based RBAC model which improves and refines the assignment of permissions to roles with awareness of the user trust and reputation. After describing the basic model, called TDRBAC for Trust-based Dynamic RBAC, we describe ways to deal with issues like privacy-preservation and delegation of roles with the consideration of user's trust.

1 INTRODUCTION

Managing access control in a computerized system can be a simple task as long as there are a few users or entities to control. However, with the growth of the number of entities within a web based system, and the exponential growth of the number of its users, there is a need to find better ways to manage access to web applications. Several access control models have been suggested over the years to simplify access control management, one of those is the Role Based Access Control(RBAC) model. RBAC is probably the most common, widely accepted and most widespread both in academic research and in actual implementation of many web systems. RBAC allows the assignment of users to roles and the assignment of roles to permissions, as opposed to direct assignment of users to permissions. However as web applications become more open (e.g. Google, Amazon, etc.) and their users are spread world-wide and are often unknown in advance, those web application get exposed to various attacks. The basic RBAC model may not be sufficient and a more refined model which assigns permissions also based on Trust is needed. In this paper, we propose such a new model called Trust-based Dynamic RBAC (TDRBAC).

Previous studies of trust-based access control such as Ray et. al (Chakraborty and Ray, 2006) change the assignment of roles to users based on their trust level. However, the permissions within a role remain the same. We believe that such assignment is too coarse and a more refined model is needed. For example two physicians may be assigned to a role of Doctor but their permissions may differ based on their trust level. In this paper we propose such a refined model called TDRBAC for Trust-based Dynamic RBAC. In TDRBAC, we incorporate traditional RBAC with awareness to the user trust level in a way that maintains the advantages of RBAC as a well-defined permission management model, and utilizing the trust level of the system entities to control permission assignment in a more refined granularly and a more dynamic way. TDRBAC reduces the need to maintain and control user to role assignment over time, by allowing flexibility and resilience to the changes in the user's trust level within the assignment of the roles to selective permissions. The model is designed to deny permissions from a user when her trust level decreases, while granting more permissions when the trust level increases.

To enhance TDRBAC usability we propose two important extensions. The first extension provides privacy, and the second one enables user rights delegation with trust awareness.

Lavi, T. and Gudes, E.
Trust-based Dynamic RBAC.
DOI: 10.5220/0005687503170324
In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016)*, pages 317-324
ISBN: 978-989-758-167-0
Copyright © 2016 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

The rest of the paper is organized as follows: In the next section we provide the background and discuss related work. In Section 3 we define and explain the basic model of TDRBAC and then we present a comprehensive example that demonstrate the usage of the proposed model. In Section 4 we discuss the extensions for privacy-preservation and user rights delegation. In Section 5 we conclude this work and discuss future work.

2 BACKGROUND AND RELATED WORK

The RBAC model was defined in 1992 by Ferraiolo et al (Ferraiolo and Kuhn, 1992), and later formalized in (Sandhu et al., 1996) by Sandhu et al, as a discretionary access control approach that is more suitable to the secure processing needs of non-military systems. RBAC was rapidly adopted both in practice and in theory studies. RBAC offers an efficient management of user's permissions, an effective enforcement of need-to-know access control principles and simplified auditing of user's permissions for regulatory compliance. On the other hand, RBAC is often considered outdated, expensive to implement and unable to accommodate real-time environmental states as access control parameters. A recent survey conducted by Fernandez et al (Condori-Fernández et al., 2012) questioned RBAC assumptions and strengths, and revealed some interesting findings. The survey shows 48% of agreement among the respondents, that changes affecting the assignment of users to roles, and roles to permissions, happen frequently and may become either an overwhelming task, or may lead to violations of the need-to-know policies in enterprise applications. We believe that within large-scale open applications that phenomena could be even more disruptive (the survey did not include open applications). Another dynamic change that RBAC itself is not sufficient to handle, is the change in the *user's trust* level over time, while it is expected that an access control system should not disregard low user trust when granting access to sensitive permissions. To overcome the lack of flexibility of RBAC, new access control models have been introduced, including Attributes Based Access Control (ABAC) (Jin et al., 2012b)(Coyne and Weil, 2013), ABAC/RBAC hybrid approach such as (EmpowerID, 2013), (Rajpoot et al., 2015) and RABAC (Jin et al., 2012a).

The idea of incorporating trust into an access control model was discussed before by several researchers. In ABAC type model, integrating trust can be achieved simply by considering the user trust as an attribute within the system (Smari et al., 2014). RBAC on the other hand is a rigid model and by that place a challenge for combining trust awareness. The access control model suggested in (Deng and Zhou, 2012), offers a set of equations to compute the user trust level in relation to a specific corresponding object. Both the user nodes and resource nodes are assigned a trust level using a trust evaluation algorithm which makes the model quite far from the simple RBAC model. The TrustBAC model, described in (Chakraborty and Ray, 2006), defines the required trust level that the user must have in order to be assigned to a specific role. If the user does not meet the required trust level of a specific role, the role assignment is denied and another role is assigned to the user. This may lead to a too simple model which may cause proliferation of roles. Another more sophisticated model was proposed by Chen and Crampton (Chen and Crampton, 2011). Their model defines quantitative factors on various components of the RBAC model and also introduce the notion of "Mitigation" that allow a specific permission to be allowed under specific requirement in case that the trust level of the user does not meet the minimum risk threshold. The model considers several mitigation strategies which can be used to compute the risk level for using a permission based on multiple paths between the user and the permission. Again this deviates from the simple RBAC structure and may impair the robust structure of RBAC and interfere with the important RBAC feature of simplified user's permission auditing.

TDRBAC on the other hand offers a way to incorporate users's trust considerations while preserving the robust skeleton of RBAC as a role-centric access control. It provides a compromise between the too simple model of TrustBAC and the complex models of (Deng and Zhou, 2012) and (Chen and Crampton, 2011). In TDRBAC, as in RBAC, the permissions explicitly define the modes of access and enable simplified auditing (Kuhn et al., 2010).

3 THE TDRBAC MODEL

3.1 TDRBAC Elements

We rely on the common definition of the following: *user*, *role*, *operation*, *object*, *permission*, and in addition lets define:

Definition 1. *Let userTrust be a real number in the range of [0, 1] that represent a user expectation to act in an honest manner as expected when 0 means that the user is expected to abuse the system and 1*

means that the user is fully trusted. userTrust is a user property.
We denote: user.userTrust $\in [0,1]$.

Definition 2. *Let permissionTrust be a real number in the range of [0, 1] that represent the minimum level of trust that a user must have in order to use the permission, in a specific role to permission assignment (see RPA definition below).*
We denote: permissionTrust $\in [1,0]$.

3.2 Assignments

Definition 3. *Let URA be user to role assignment. a user can be assigned to many roles in many-to-many relation.*
We denote: $URA \subseteq USERS \times ROLES$.

Table 1: URA assignment example.

USER	ROLE
Mike	Manager
Joe	Guest
Lisa	Admin

Definition 4. *Let RPA be role to permission assignment. A role can be assigned to many permissions in a many-to-many relation. Each role to permission assignment has a trust level.*
We denote: $RPA \subseteq ROLES \times PERMISSION \times [0,1]$.
The range [0,1] represent the RPA trust level which represent the minimum trust level that the user must have in order use this permission in this role.

Table 2: RPA Assignment Example.

ROLE	PERMISSION	Trust
Manager	Can assign roles to users	0.9
Guest	Read public posts	0
Admin	Change system configurations	1

3.3 How TDRBAC Works

When a user requests to perform some operation on a system resource, the Authorization Center (AC) verifies that the user is assigned to a role with the corresponding permission. If the user is not assigned to any role with a corresponding permission, the request is rejected. If a corresponding permission was found, the AC will consider the *RPA* trust level: If the *RPA* trust level is 0, then the request is granted and there is no need to calculate the user trust. Otherwise, the AC will calculate the user trust and allow the request only if the user trust level is greater or equal to the *RPA* trust level.

3.4 Dealing with Collisions

In some cases, users may be assigned to more than one role in a specific session. For example, a sales person can also be a board member and get assigned to two different roles. The common approach to handle multiple roles for each user is to restrict the user to select specific role for each new session. For example, the user must define whether she is acting as a ”board member” or as a ”sales person” once the session is created. That leads to a massive overhead on system usage and while it’s expected on high sensitive systems, it could be very interfering in normal systems. Therefore we believe that the model should handle a multiple roles assignments in a specific session.

A collision is a situation when a specific permission is restricted from the user in one RPA when the *RPA.trustLevel* is greater than the *userTrust* level, and it is granted by another *RPA* when the *RPA.trustLevel* is lesser than the *userTrust* level. In a case of collision, the decision to grant or reject the user’s request is based on a predefined security policy. The Security Officer can decide to reject the permission if there is at least one *RPA* with a greater corresponding *RPA.trust* than the *userTrust*, or she can decide to accept any permission request once there is at least one RPA with a lesser corresponding *RPA.trust* than the *userTrust*.

The selected policy should be considered very carefully. The simple-minded approach would be to allow the permission if there is at least one corresponding *RPA* that allows it, but later in this paper we show that this approach may provoke a security breach by allowing the user a subset of permissions that the user could use to induce an attack or abuse the system even with a low level of trust.

3.5 TDRBAC in Action (Detailed Scenario)

In order to demonstrate TDRBAC usage we will show an access control policy for a technical customer support system. To leverage the flexibility of TDRBAC we want to have a small number of roles and let the trust level to allow specific permission in the limits of the role. In our system, we have only three roles:

- Customer
- Support agent
- Administrator

A *customer* can create issues (a case ticket) to describe a problem, a bug or request and close the ticket once it is resolved. Once the user gain more trust we

can assign more permission to the user, like adding files to the issue.

An *agent* can be assigned to an issue, view and instruct the customer to resolve it. As the agent gains more trust, we can assign advanced capabilities like control the customer desktop in order to resolve issues.

An *administrator* can change system properties and configuration and manage users and roles. In our system, the user "root" is always assign to the administrator role and is always fully trusted.

Another capability that we have in our support system is the "Knowledge-Base". The KB is a set of articles with known bugs and issues that the agents can use to share resolutions with each other. Once an agent gets an issue for which he has no solution, he can browse the KB to search for a solution. A trusted customer can also get access to the KB, and trusted agent can add or edit articles in the KB.

The complete set of permissions is as follows:

- A customer can:
  - Create a new issue
  - Add comments to own issues
  - Add files (like logs, screenshots etc.)
  - Close own issues
  - Collaborate on issues of other users
  - Browse the knowledge-base
  - Can create more than one issue in 24 hours (to avoid spam)
- An agent can:
  - Take ownership on an issue
  - View customer desktop
  - Gain control on customer desktop and files system
  - Resolve an issue
  - Add comments to issues
  - Assign issues to other agents
  - Add article to the knowledge-base
  - Edit articles in the knowledge-base
  - Delete article on the knowledge-base
  - Gain control on customer desktop and files
- An administrator can:
  - Change system configuration
  - Register new users
  - Manage users details
  - Manage user's roles

The next step is to assign a trust level to each permission. The most basic capabilities, which any new user must have, will be assigned to a trust level of 0. As the permission is more advanced, we assign a higher trust. New users in our system starts with a trust level of 0.

Table 3 list the final RPA of our system:

Table 3: RPA Assignment.

ROLE	PERMISSION	Trust
Customer	Create a new issue	0
	Add comments to own issues	0
	Close own issues	0
	Browse the KB	0.25
	Create more than one issue in 24h	0.25
	Add files to an issue	0.75
	Collaborate on issues of other users	1
Agent	Resolve an issue	0
	Add comments to issues	0
	Add files to an issue	0.25
	Add article to the KB	0.25
	Assign issues to other agents	0.5
	Edit articles in the KB	0.5
	Take ownership on an issue	0.75
	View customer desktop	0.75
	Delete article on the KB	0.75
	Control on customer desktop/files	1
Admin	Register new users	0.25
	Manage users details	0.75
	Change system configuration	1
	Manage user's roles	1

As is shown in the example above, our model of access control allows managing just a few number of roles, and yet getting the flexibility and dynamic permission assignment that by other access control models will most likely require more roles to manage such as: basic customer and advanced customer or new agent and veteran agent. Nevertheless, a user cannot gain any permission that is not part of her roles.

Permissions that are essential for the role are set with the $RPA_t rust$ of 0, that is to insure that users get the minimum required permission for the role they are assigned to. As the users gain more trust, they also gain access to more sensitive or strong permissions. The problem of assigning trust level to permissions is dealt in more detail in Section 3.7.

3.6 Calculating *userTrust*

TDRBAC does not restrict, nor dictate, the way of calculating the user trust level. Many models suggest

a way of calculating a user trust level based on the user history, experience and recommendation, see for example (Ray and Chakraborty, 2004). Yet, it is expected that trust level of a user would be calculated as close as possible to runtime, each time that the user is asking to use a permission. To improve performance the trust calculation function can use cached data and avoid re-calculation of user trust in each access operation. (For example, the user trust calculation can use the same value for the next 5 minutes, assuming that in 5 minutes there would be no major change in the user trust level.) The system administrator must be aware that a malicious user can exploit the time difference of user trust calculation to perform illegal actions before losing her permissions due to the loss of trust. Note that the trust model may also take into account the number of times the user was denied access to permissions. If this number is high it may indicates an intrusion attempt by a low trust user.

A related issue is the issue of dynamic enforcement. When the trust level decreases, one loses access but what happens to the accesses already performed? In a transaction system (like a Database transaction) one may require that the entire transaction be performed with a single trust level, if such a loss of trust has happened, the entire transaction should be aborted.

3.7 Computing *Trust* of Role to Permission Assignment

There are two key components in our model: One is the calculation of the user trust level. Obviously, if the user's trust is not genuine, the complete model looses it's credibility. Yet, as previously explained, TDRBAC does not dictate the way of calculating the user trust level. The second is the trust level that is chosen for each *Role* to *Permission* Assignment (*RPA*). Choosing high level of trust for each permission can lead to a low usability while too low trust could lead to an untrustworthy and vulnerable system. Proper assignment of *RPA* should balance between usability and security. To improve system usability, we should consider to assign the lower trust level possible, hence having the permissions accessible to as many users as possible. On the other hand we must prevent low-trusted users to perform dangerous or sensitive operations that can cause damage to the system.

The definition of the RPA can be done by the security administrator based on accumulated expertise and experience, however a more systematic methodology is desired. Next we propose such methodology based on information collected during system operation. In order to formulate the process of permissions trust assignment we need to add the following definitions:

Definition 5. *Let Incident be a scenario where a user perform a set of operations that leads to a security breach, as system shutdown, data loss or data manipulation.*

Definition 6. *Let* $Damage(Incident)$ *be a real number in the range of* $[0,1]$ *that represents the projection of an Incident on the system, when* 0 *means that the incident does not effect the system, and* 1 *means a complete destruction or system breakdown. In a commercial system, the Damage should be calculated by the financial loss, and the cost to repair the damage. The value of Damage should be normalized by the total cost that the organization can handle. In a non-commercial system (such as military system) the value of Damage reflect the damage of an incident based on risk assessment.*

In the process described below we use the *Damage* as highly correlated with the required Trust level.

Definition 7. *Let* $common(permission)$ *be a real number in the range of* $[0,1]$ *that represent the usage probability of a specific permission within the system. A more commonly used permission should be assigned to a lower value of trust as possible while less common permission, with requirement for a high level of trust would have a smaller effect on the overall system usability.*

In practice Common may be learned from the system logs.

Definition 8. *Let* $PER(Incident)$ *be a subset of permissions, that are needed by a user to cause an Incident.*

We Denote: $Per(Incident) \subseteq PERMISSIONS$

With the above definitions we look for a procedure that will assign an "optimal" match of trust levels to permissions. One may define the optimality criteria in different ways, but in general, the trust of a permission shuld increase with the incident damage it was involved with, and decreases with its commonality. The tradeoff between these two factors is to be determined by the security administrator. While we have not fully investigated the complexity of the algorithm to solve this optimality problem, below we suggest a simple and practical heuristic for the trust assignment. The procedure is as follows:

1. Sort all *Incidents* by the value of the damage in descending order, such that *Incident* with high value of damage is ordered before an *Incident* with a lower level of damage.
2. Sort all *permissions* by the value of *common* in ascending order, such that *permission* that is less

usable is ordered before a more commonly used permission.

3. initialize all permissions to a minimal trust value (as determined by the security administrator)
4. Set $i = 0$
5. For the subset of permissions $PER(Incident_i)$ if the there is no permission with a trust level of $Damage(Incident_i)$ or higher, than assign trust level of $Damage(Incident_i)$ to the first permission in the subset of permission (in the "common" order).
6. adjust i in 1 and repeat step 5.

Note that the procedure above insures that each incident will cause at least one permission, usually the less common one, to acquire a high trust level, while the rest may stay in their lower trust levels.

4 TDRBAC EXTENSIONS

In the following section we propose two important extensions for the basic TDRBAC model:

- TDRBAC-P - extension for privacy-preservation
- TDRBAC-D - extension for delegation.

4.1 TDRBAC-P Extension for Privacy-preservation

Privacy-preservation issues are becoming more and more important these days as social networks and large open systems raise crucial questions regarding the usage of the user's private data. Here we describe an extension to TDRBAC dealing with the privacy issue.

To address the concerns of privacy-preservation in TDRBAC we incorporate the broadly used notion of Purpose as proposed in (Yang et al., 2007). We define purpose as follows:

Definition 9. *Let Purpose be a reason for data collection and data access. The purpose is "well-defined", that is to say that the purpose is unambiguous in the model implementation. Purposes cannot overlap or partially overlap.*
Let PUR be the collection of purposes in the access control policy.

We define Pi as a set of purposes: $Pi = \{p1, p2, p3, \ldots\} | p \in PUR$

We extend the RPA assignment to include the allowed purposes:

Definition 10. $RPA \subseteq ROLES \times PERMISSIONS \times PUR \times [0,1]$

With each permission request, the system must specify the purposes for which the permission is requested. The permission request would be in the form:
$f(user, permission, purpose)$.
This definition enables the specification of different trust levels for different purposes for the same (role,permission) pair. An example for an RPA assignment including purposes is shown in table 4

In case the trust level is not sufficient for a specific purpose, the request may be denied, or less private data for a lower purpose may be returned. One useful way of dealing with purposes in case of a of Read access, is to allow two types of answers to a Read request:

- **Abstract Data** - a modified version of the records without any private data (such as: users identity, medical details, students grades, etc.) Another form of Abstract data may be a k-anonymized data such as in (Bayardo and Agrawal, 2005).
- **Detailed Data** - The full detailed data as requested by the user.

The way the AC behaves in this case is dependent on the Privacy policy. If both the purpose and trust level match, the access is permitted. If the purpose match and the trust level is too low, the request may be denied or the data used for a lower purpose may be returned, if the trust level for the lower purpose is sufficient.

Usage Example

A doctor is requesting read access to patient lab tests. The request purpose can be one of:

- $p1$ - For research purposes (low purpose)
- $p2$ - To write a prescription (high purpose)

The *RPA* for lab results are as specified in table 5.

The returned results in case of $p1$ is abstract data (e.g - without the patient personal details). The results in case of $p2$ would be complete lab results. If the doctor has a trust level of 0.4 and she request the "write prescription" purpose, the system may return only abstract data (lower the purpose) or deny the request.

4.1.1 Purpose Enforcement

When implementing the approach of purpose, the system must enforce the purpose notion to avoid manipulation of the permission request (Colombo and Ferrari, 2014). To achieve purpose enforcement, the system should identify the actual use of the permission in the time of the request and substitute the purpose within the permission request. In the above example,

Table 4: RPA Assignment with Purposes.

Role	Permission	Purposes	Trust
Administrative Assistant	Read contacts details	Schedule meetings	0.5
Engineer	Access previous studies and researches	Resolve system flaws	0.5
CFO	Access business plans	Create budget plans	0.75

Table 5: RPA Assignment with Purposes.

Role	Permission	Purposes	Trust
Doctor	Read lab results	Write	0.5
Doctor	Read lab results	prescription	
Doctor	Read lab results	Research	0.3

when the doctor request permission to review lab results, the system should identify whether the doctor is currently writing prescription to a patient and substitute the purpose accordingly.

4.2 TDRBAC-D Extension for Delegation

Delegation is the assignment of a role to a user. Normally, role assignments are performed by the security officer or administrators ("Administrative Delegation"). However, sometimes we want to allow one user to delegate his own roles to another user ("User Delegation"). Administrative delegations are usually permanent, while user delegations are usually temporary. From now, we refer to "delegation" as user delegation.

The ability to delegate user roles to other users is very important in large-scale open systems. By letting the users to delegate their own roles, with the proper limitation and trust awareness, we simplify the management of the user roles, and avoid too many administrative changes. Examples for constraints which can be used in roles delegation are given by Crampton et al in (Crampton and Khambhammettu, 2008).

To enable delegation which is Trust dependent, we extend the basic model of TDRBAC as follows:

4.2.1 Definitions

Definition 11. *Let delegator be a user who perform role delegation to another user.*

Definition 12. *Let delegatee be a user who receive role delegation from another user*

Definition 13. *Let delegationThreshold be a real number in the range [0, 1]. The number represents the minimum trust level that the user (the delegator) must have in order to perform delegation of a specific role to another user. We assign a delegationThreshold to any role that we allow users to delegate. A role that is not allowed for delegation does not have a delegationThreshold value. We denote: delegationThreshold* $\in [0,1]$

Definition 14. *Let delegatedTrust be a calculated value based on the user trust level and the delegator trust. When the user request a permission based on a delegated role, we use the delegatedTrust value instead of the userTrust.*
Formula:
$delegatedTrust = Delegator.userTrust \times Delegatee.usertTrust$

Definition 15. *Let TRA be a delegationThreshold to role assignment. An assignment of a trust level threshold for any role that is eligible for delegation. A role is eligible for delegation only if the delegator assigned to the role and has a trust level that is greater or equals to the delegated threshold of the specific role. We denote:* $TRA \subseteq role \times delegationThreshold$

Table 6 demonstrate TRA assignment

Table 6: TRA Assignment.

Role	Delegation Threshold
Engineer	0.5
Director	0.8
Salesperson	0.6

Definition 16. *Let DRA be delegated roles assignment - A list of delegated roles performed by delegators to delegatees. We denote:* $DRA \subseteq USER \times ROLE \times USER$

Table 7 demonstrate DRA assignment

Table 7: DRA Assignment.

Delegator	Role	Delegatee
John	Engineer	Bob
Michael	Director	Lisa
Alice	Salesperson	Anna

4.2.2 Understanding TDRBAC-D

When a user requests a permission, the AC will first runs the basic TDRBAC function. If a direct *user* to *role* assignment does not allow the permission requested, the AC will run the delegation permission functions by listing the delegated roles assignment where the *user* is mentioned as Delegatee. Than the

AC will verify that the assignment is valid. A valid delegation must satisfy the following conditions:

1. The delegator is assigned to the role that is being delegated.
2. The role is eligible for delegation.
3. The delegator trust level is greater or equal to the *delegationThreshold*.

If the requested permission is listed in one of the valid roles, than the permission is granted if the computed *delegatedTrust* is greater or equals to the *RPA* trust level.

5 CONCLUSIONS

The concept of Trust Aware role based access control model was recognized in previous work. The challenge in this work was to preserve the strengths of the well known RBAC model as a role-centric access control. The model presented here shows a middle way in that it provides a refined enough level of trust awareness based on permissions, yet it is simple, enables simplified auditing and can be easily understood and enforced. In addition to the basic model, two extensions were presented, one for Privacy purposes, the other for Delegation purposes.

In future work we like to investigate the RPA computation problem we described in section 3.7 further and evaluate it under various simulation conditions. We also plan to combine the model extensions and test the model in a real-life scenario of a large company.

REFERENCES

Bayardo, R. J. and Agrawal, R. (2005). Data privacy through optimal k-anonymization. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 217–228. IEEE.

Chakraborty, S. and Ray, I. (2006). Trustbac: integrating trust relationships into the rbac model for access control in open systems. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 49–58. ACM.

Chen, L. and Crampton, J. (2011). Risk-aware role-based access control. In *Security and Trust Management - 7th International Workshop, STM 2011, Copenhagen, Denmark, June 27-28, 2011, Revised Selected Papers*, pages 140–156.

Colombo, P. and Ferrari, E. (2014). Enforcement of purpose based access control within relational database management systems. *IEEE Trans. Knowl. Data Eng.*, 26(11):2703–2716.

Condori-Fernández, N., Franqueira, V. N., and Wieringa, R. (2012). Report on the survey of role-based access control (rbac) in practice.

Coyne, E. and Weil, T. R. (2013). Abac and rbac: Scalable, flexible, and auditable access management. *IT Professional*, 15(3):14–16.

Crampton, J. and Khambhammettu, H. (2008). Delegation in role-based access control. *Int. J. Inf. Sec.*, 7(2):123–136.

Deng, W. and Zhou, Z. (2012). A flexible rbac model based on trust in open system. In *Intelligent Systems (GCIS), 2012 Third Global Congress on*, pages 400–404.

EmpowerID, w. p. (2013). Best practices in enterprise authorization: The rbac/abac hybrid approach.

Ferraiolo, D. and Kuhn, R. (1992). Role-based access control. In *In 15th NIST-NCSC National Computer Security Conference*, pages 554–563.

Jin, X., Krishnan, R., and Sandhu, R. (2012a). A role-based administration model for attributes. In *Proceedings of the First International Workshop on Secure and Resilient Architectures and Systems*, pages 7–12. ACM.

Jin, X., Krishnan, R., and Sandhu, R. S. (2012b). A unified attribute-based access control model covering dac, mac and rbac. *DBSec*, 12:41–55.

Kuhn, D. R., Coyne, E. J., and Weil, T. R. (2010). Adding attributes to role-based access control. *Computer*, 43(6):79–81.

Rajpoot, Q. M., Jensen, C. D., and Krishnan, R. (2015). Integrating attributes into role-based access control. In *Data and Applications Security and Privacy XXIX*, pages 242–249. Springer.

Ray, I. and Chakraborty, S. (2004). A vector model of trust for developing trustworthy systems. In *In European Symposium on Research in Computer Security , Sophia Antipolis (France*, pages 260–275. Springer-Verlag.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-based access control models. *Computer*, 29(2):38–47.

Smari, W. W., Clemente, P., and Lalande, J. (2014). An extended attribute based access control model with trust and privacy: Application to a collaborative crisis management system. *Future Generation Comp. Syst.*, 31:147–168.

Yang, N., Barringer, H., and Zhang, N. (2007). A purpose-based access control model. In *Proceedings of the Third International Symposium on Information Assurance and Security, IAS 2007, August 29-31, 2007, Manchester, United Kingdom*, pages 143–148.